# MODELLING AND CONTROLLING A BIO-INSPIRED FLAPPING-WING MICRO AERIAL VEHICLE

A Thesis
Presented to
The Academic Faculty

by

David E. Smith

In Partial Fulfillment
of the Requirements for the Degree
Masters of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2011

# MODELLING AND CONTROLLING A BIO-INSPIRED FLAPPING-WING MICRO AERIAL VEHICLE

Approved by:

Professor George Vachtsevanos
School of Electrical and Computer Engineering
*Georgia Institute of Technology*

Professor David Taylor
School of Electrical and Computer Engineering
*Georgia Institute of Technology*

Professor Ayanna Howard
School of Electrical and Computer Engineering
*Georgia Institute of Technology*

Date Approved: 9 January 2012

# ACKNOWLEDGEMENTS

I am grateful to Dr. George Vachtsevanos for the research opportunities he provided me and to Dr. Jayant Ratti who encouraged and exemplified self-motivation. I also want to thank my wife, Sarah, for her support and encouragement in pursuing this research as in all my interests.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# SUMMARY

The objective of this research is to verify the three degree of freedom capabilities of a bio-inspired quad flapping-wing micro aerial vehicle in simulation and in hardware. The simulation employs a nonlinear plant model and input-output feedback linearization controller to verify the three degree of freedom capabilities of the vehicle. The hardware is a carbon fiber test bench with four flapping wings and an embedded avionics system which is controlled via a PD linear controller. Verification of the three degree of freedom capabilities of the quad flapping-wing concept is achieved by analyzing the response of both the simulation and test bench to pitch, roll, and yaw attitude commands.

# CHAPTER I

# INTRODUCTION

## 1.1 State of the Art in Micro UAVs

Academia, industry, and the military have expended considerable research and development on large and medium-scale Unmanned Aerial Vehices (UAVs) which range in wingspan from one to several dozen feet. In the last decade research has turned toward developing Micro Aerial Vehicles (MAVs), which DARPA originally defined as having less than a 16 cm wingspan and weighing less than 100 grams.

As UAV research has moved into the scale of MAVs, many problems have been encountered in developing efficient flight mechanisms. A source of inspiration and innovation for novel MAV flight mechanisms has come from the study of insects and their mechanisms for efficient flight. One such mechanism is flapping wings and, over the last decade, several flapping wing MAVs have been developed. One of the most successful in terms of agility is AeroVironment's Nano Hummingbird [1]. However, the Nano Hummingbird's usefulness is limited by its eleven minute flight time.

One subset of the engineering challenges in developing a flapping wing MAV is in modeling and designing a controller for the vehicle. Nonlinear models and controllers for quad rotors have been developed at the University of Aalbork, Instituto Superior Technico, as well as others [6, 2, 3]. Models for flapping wing mechanisms have been developed at the University of Delaware, University of California at Berkeley, and others [5, 8]. While many have performed research on UAVs with four actuators, little research has been on control system implemention on MAVs with four flapping wings.

## 1.2 Project History

This masters project is part of a larger project in the Intelligent Controls Systems Laboratory to develop a quad flapping-wing (QFW) micro aerial vehicle (MAV). It has been theoretically shown that a craft with four flapping wings has 50% higher efficiency than a craft with two flapping wings [7]. In an effort to create a MAV with long flight endurance, our research team has developed a four flapping wing MAV inspired by the dragonfly. Progress on the project has included theoretical calculations on the efficiency increase in a QFW MAV, mechanical and electrical design and construction of a QFW MAV test-bench, and a linear controller simulation for the QFW MAV [7]. Using this test bench, this research will verify the three rotational degree of freedom capabilities of the QFW concept.

## 1.3 Flapping-Wing Micro Aerial Vehicle Operation

The four wing design uses a novel maneuvering scheme as shown in Figure 1.1. Each Euler motion is achieved by producing a thrust differential across opposite pairs of wings. Pitch motion is achieved via a thrust differential across the front and rear wings, roll across the left and right wings, and yaw across diagonal wings (since each wing also produces a horizontal thrust component). Lift is generated by the sum of the wing thrusts in the direction opposite the force of gravity.

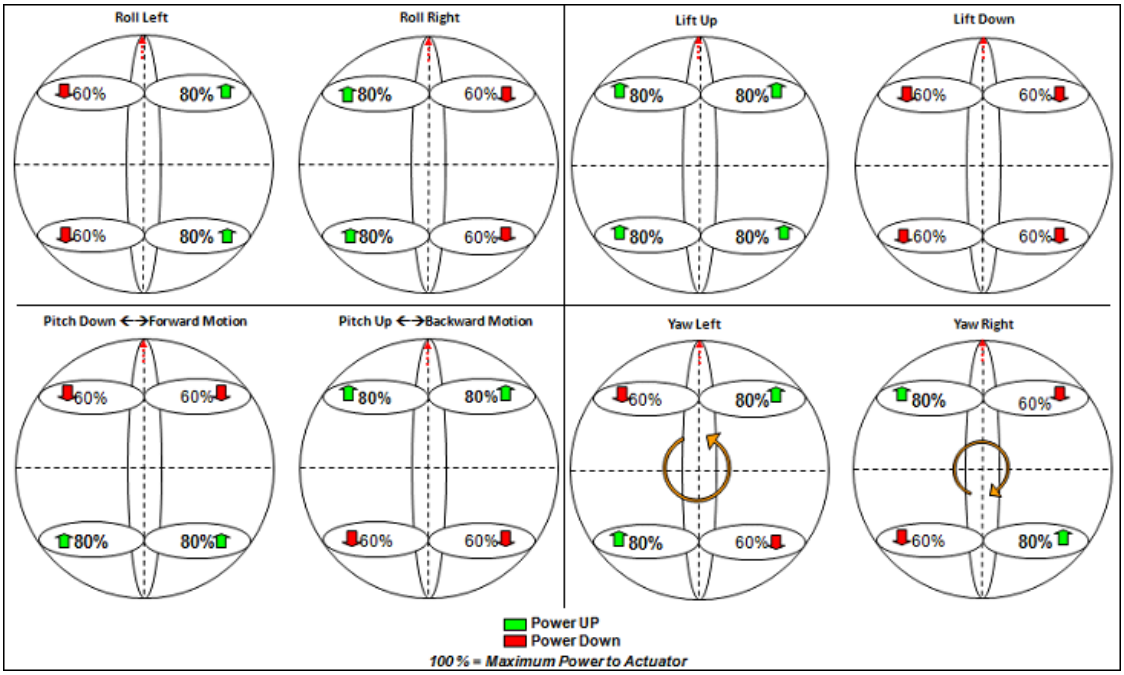



Figure 1.1: Flight Manuevers Acheived via Thrust Differentials [7]

# CHAPTER II

# MODELLING

## 2.1 Definitions

The motion of the MAV can be described using two reference frames, the inertial frame and the body frame. Each frame consists of a point in space labeled the origin and three orthonormal vectors constituting the basis. We define the inertial frame to be the frame $F_I = \{O_I, \overrightarrow{i_I}, \overrightarrow{j_I}, \overrightarrow{k_I}\}$ whose origin, $O_I$, lies on the surface of the earth and whose basis set $\{\overrightarrow{i_I}, \overrightarrow{j_I}, \overrightarrow{k_I}\}$ follows the North-East-Down convention. We define the body frame to be the frame $F_B = \{O_B, \overrightarrow{i_B}, \overrightarrow{j_B}, \overrightarrow{k_B}\}$ whose origin, $O_B$, lies at the center of gravity of the MAV and where $\overrightarrow{i_B}$ points out the front of the MAV, $\overrightarrow{j_B}$ points starboard, and $\overrightarrow{k_B}$ points out the bottom of the MAV. These conventions, with $\overrightarrow{k} = \overrightarrow{i} \times \overrightarrow{j}$, provide a right-handed system which is the assumed Cartesian coordinate system in physical laws and vector mathematics. We define rotations about each axis to be positive right-handed rotations for both the inertial and body frames as shown in Figure 2.1.

In the body frame, we define linear velocity to be $v_B = \{u, v, w\}$ and rotational velocity to be $\omega_B = \{p, q, r\}$. Linear and angular position in the body frame have no meaning as the origin of the body frame is fixed to the center of gravity of the vehicle and the basis vectors are relative to the vehicle's orientation.

In the inertial frame, linear position is described as $p = \{x, y, z\}$ and linear velocity is given as $v_I = \{v_x, v_y, v_z\}$. Rotational position is defined using Z-Y-X Euler angles with $\theta$, $\phi$, $\psi$ (roll, pitch, yaw) being right-handed rotations. Linear accelerations in the inertial frame are represented by the derivatives $\dot{v}_I = \{\ddot{x}, \ddot{y}, \ddot{z}\}$.

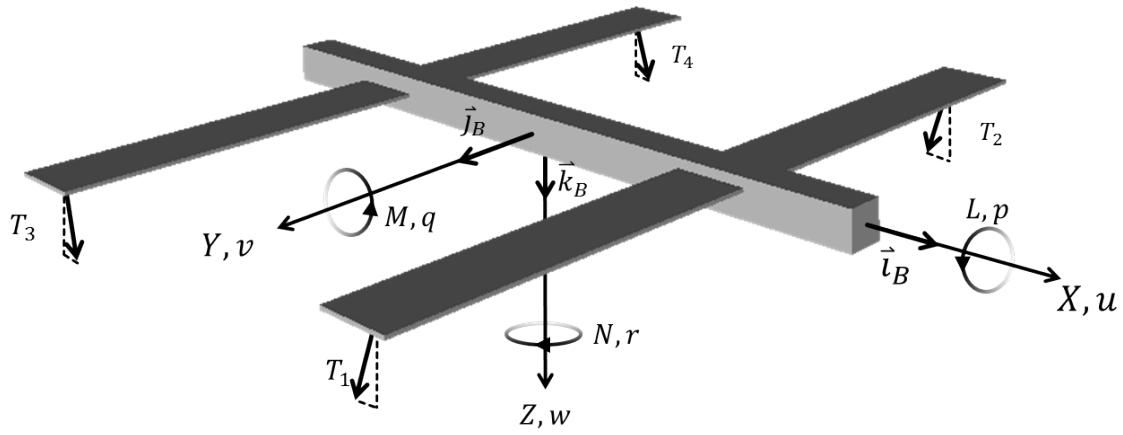



Figure 2.1: Definition of Variables and Axes

## 2.2 System Dynamics

Our goal is to derive a nonlinear plant model in the form $\dot{x} = f(x) + g(x)u$, where $x$ is our state vector and $u$ is our input vector. The size of our state vector which defines our object's configuration is at least equal to twice the number of degrees of freedom in our system. We will model the MAV as a rigid body possessing six degrees of freedom, three rotational and three translational.

Our approach is to formulate the translational and rotational equations of motion for the MAV in the body frame and then to transform the resulting linear and angular velocities to the inertial frame. This method is useful because the inertia matrix is constant with reference to the body frame since the MAV is a rigid body. We avoid representing the inertial matrix for the rotating body in the inertia frame which would result in a time-varying matrix.

### 2.2.1 MAV Equations of Motion

We can only directly apply Newton's laws to the inertial frame since, by definition, an inertial frame is one in which Newton's laws hold. After doing so, we can then derive the equations of motion as observed in the body frame which we expect will depend on the acceleration of the frame and will be supplemented by fictitious forces such as the coriolis, centrifugal, or Euler forces. In the inertial frame, Newton's second laws for translational and rotational motion of the MAV are, respectively,

$$\overrightarrow{f} = m\,\overrightarrow{\dot{v}}_I \tag{2.1}$$

$$\overrightarrow{\tau} = I\,\overrightarrow{\dot{\omega}}_I \tag{2.2}$$

The terms $\left.\frac{d\overrightarrow{v}}{dt}\right|_I$ and $\left.\frac{d\overrightarrow{\omega}}{dt}\right|_I$ are the time derivatives of the vectors $v_I$ and $\omega_I$ as viewed by an observer in the inertial frame and we need to express these in terms of vectors defined in the body frame. The velocity vector $\overrightarrow{v}$ is a free vector which the observers in both the inertial and body frames can represent in their own basis sets as $v_I$ and $v_B$, respectively. While a simple coordinate transformation between the two frames can verify to each observer that both $v_I$ and $v_B$ point in the same direction, the two observers would not agree on the rate of change of $\overrightarrow{v}$ since each frame is rotating with respect to the other. In the inertial frame, the observer views the absolute rate of change of $\overrightarrow{v}$ and can formulate it in terms of the rotating body frame's basis set as:

$$\overrightarrow{\dot{v}}_I = \dot{v}_1\overrightarrow{i}_B + \dot{v}_2\overrightarrow{j}_B + \dot{v}_3\overrightarrow{k}_B + v_1\overrightarrow{\dot{i}_B} + v_2\overrightarrow{\dot{j}_B} + v_3\overrightarrow{\dot{k}_B} \tag{2.3}$$

However, since from the body frame perspective, the basis vectors $\{\overrightarrow{i_B}, \overrightarrow{j_B}, \overrightarrow{k_B}\}$ are constant, the observer in the body frame formulates the rate of change of $\overrightarrow{v}$ to be:

$$\overrightarrow{\dot{v}}_B = \dot{v}_1\overrightarrow{i}_B + \dot{v}_2\overrightarrow{j}_B + \dot{v}_3\overrightarrow{k}_B \tag{2.4}$$

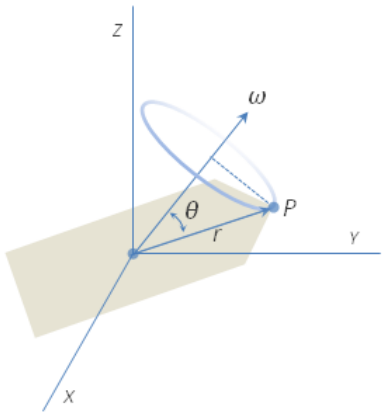



Figure 2.2: Vector in Rotating Body Frame

From the perspective of the inertial frame, the rate of change of the basis vectors $\{\dot{\vec{i}}_B, \dot{\vec{j}}_B, \dot{\vec{k}}_B\}$ of the body frame rotating with angular velocity $\omega$ is:

$$\{\dot{\vec{i}}_B, \dot{\vec{j}}_B, \dot{\vec{k}}_B\} = \{\omega \times \vec{i}_B, \omega \times \vec{j}_B, \omega \times \vec{k}_B\} \tag{2.5}$$

To show this, consider our rigid body which is rotating with absolute angular velocity $\omega$ with respect to an inertial frame XYZ as shown in Figure 2.2. Let us calculate the absolute velocity $v$ of a point $P$ in the body as it moves along the circle about $\vec{\omega}$, the instantaneous axis of rotation. Since the radius of the circle is $r\sin\theta$, the speed of $P$ is $\dot{s} = \omega r \sin\theta$, where $s$ is the displacement along the circle, $r$ is the magnitude of the position vector $\vec{r}$ from the origin to $P$, and $\theta$ is the angle between the instantaneous axis of rotation and the position vector $\vec{r}$. Because the direction of the velocity of $P$ is tangential to the circle, we can write:

$$\vec{v} = \vec{\omega} \times \vec{r} \tag{2.6}$$

Since the basis vectors in the body frame behave as position vectors in a rigid body, we can substitute equation (2.5) into equation (2.3) to get the absolute rate of change of $\vec{v}$ in

terms of measurable vectors in the body frame as shown below:

$$\dot{\vec{v}}_I = \dot{v}_1 \dot{\vec{i}}_B + \dot{v}_2 \dot{\vec{j}}_B + \dot{v}_3 \dot{\vec{k}}_B + v_1 \omega \times \vec{i_B} + v_2 \omega \times \vec{j_B} + v_3 \omega \times \vec{k_B} \tag{2.7}$$

Combining and simplifying equations (2.4) and (2.7), we get:

$$\dot{\vec{v}}_I = \dot{\vec{v}}_B + \omega \times \vec{v} \tag{2.8}$$

Although the inertial frame is fixed and the body frame is rotating, we can generalize this equation to apply to any two frames $A$ and $B$ which may be rotating with respect to each other and neither is more fundamental than the other. Note also that $\vec{v}$ can be any free vector $\vec{n}$. As a result, the equation is symmetric and valid in both forms below:

$$\dot{\vec{n}}_A = \dot{\vec{n}}_B + \omega_{BA} \times \vec{n} \tag{2.9}$$

$$\dot{\vec{n}}_B = \dot{\vec{n}}_A + \omega_{AB} \times \vec{n} \tag{2.10}$$

In these equations, $\omega_{BA}$ is the rotation of frame $B$ as viewed from frame $A$ and vice versa. Since each frame views all rotations as a rotation in the other frame, $\omega_{BA} = -\omega_{AB}$. Recall that $\vec{n}$ is expressed in the body frame basis set and is the same for both frames. Since the generalized equations (2.9) and (2.10) are valid for both vectors $\vec{v}$ and $\vec{\omega}$ in equations (2.1) and (2.2), using a substitution, we can write:

$$\vec{f} = m \dot{\vec{v}}_B + \omega_{BI} \times m \vec{v} \tag{2.11}$$

$$\vec{\tau} = I \dot{\vec{\omega}}_B + \omega_{BI} \times I \vec{\omega} \tag{2.12}$$

Note that (2.11) and (2.12) are the equations of motion with respect to the inertial frame expressed in the body frame basis set.

### 2.2.2 Kinematic Equations

Recall, that our goal is to derive a nonlinear plant model in the form $\dot{x} = f(x) + g(x)u$ and our approach is to formulate the translational and rotational equations of motion for the MAV in the body frame and then to transform the resulting linear and angular velocities to the inertial frame. Now that we've formulated the equations of motion in the body frame, we need to find the inertial frame velocities $v_I$and $\omega_I$.

Recalling our notational definitions for $\overrightarrow{v}_B$ and $\overrightarrow{\omega}_B$, we can rearrange and present equation (2.11) in matrix form as:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} - \begin{bmatrix} qw - rv \\ ru - pw \\ pv - qu \end{bmatrix} \tag{2.13}$$

In the body frame, the net translational force is the sum of wing thrusts, $f_T$, and gravity, $f_G$. The wing thrusts, $\{T_1, T_2, T_3, T_4\}$, are always directed in the negative z direction of the body frame and so are written $f_T^B = \begin{bmatrix} 0 & 0 & -(T_1 + T_2 + T_3 + T_4) \end{bmatrix}^T$. The force of gravity is always directed in the positive z direction of the inertial frame, $f_G^I = \begin{bmatrix} 0 & 0 & mG \end{bmatrix}$. In order to express $f_G^I$ in the body frame we use a Z-Y-X Euler angle rotation matrix. The rotation matrix is composed using three rotations. First, a rotation, $R(\psi)$, of the inertial frame vector $\overrightarrow{n}^I = \left\{ n_i^I, n_j^I, n_k^I \right\}$ about the body frame z-axis, $\overrightarrow{k_B}$, by a yaw, $\psi$, giving:

$$\begin{bmatrix} n_i^1 \\ n_j^1 \\ n_k^1 \end{bmatrix} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_i^I \\ n_j^I \\ n_k^I \end{bmatrix} \tag{2.14}$$

This rotation creates a intermediate frame, 'frame 1', with the origin remaining at the body frame center of gravity and the basis vectors being $\left\{ \overrightarrow{i}_1, \overrightarrow{j}_1, \overrightarrow{k}_B \right\}$. Note the z-axis basis

vector remains the same. We then perform a rotation, $R(\phi)$, of the new vector, $\overrightarrow{n}^1$, about the new y-axis, $\overrightarrow{j_1}$, by a pitch, $\phi$:

$$\begin{bmatrix} n_i^2 \\ n_j^2 \\ n_k^2 \end{bmatrix} = \begin{bmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{bmatrix} \begin{bmatrix} n_i^1 \\ n_j^1 \\ n_k^1 \end{bmatrix} \tag{2.15}$$

The second intermediate frame, 'frame 2', has the basis vectors $\left\{ \overrightarrow{i}_2, \overrightarrow{j}_1, \overrightarrow{k}_2 \right\}$, since the y-axis remains the same. Lastly, we perform a rotation, $R(\theta)$, of $\overrightarrow{n}^2$, about the new x-axis, $\overrightarrow{i_2}$, by a roll, $\theta$:

$$\begin{bmatrix} n_i^B \\ n_j^B \\ n_k^B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} n_i^2 \\ n_j^2 \\ n_k^2 \end{bmatrix} \tag{2.16}$$

By multiplying these matrices together we combine all three rotations into one rotation matrix with which we can express a vector represented in the inertial frame basis set in terms of the body frame basis set, as shown below:

$$\begin{bmatrix} n_i^B \\ n_j^B \\ n_k^B \end{bmatrix} = \begin{bmatrix} c\phi c\psi & c\phi s\psi & -s\phi \\ s\theta s\phi c\psi - c\theta s\psi & s\theta s\phi s\psi + c\theta c\psi & s\theta c\phi \\ c\theta s\phi c\psi + s\theta s\psi & c\theta s\phi s\psi - s\theta c\psi & c\theta c\phi \end{bmatrix} \begin{bmatrix} n_i^I \\ n_j^I \\ n_k^I \end{bmatrix} \tag{2.17}$$

Note that we can describe a vector in the body frame in terms of the inertial frame basis set by applying the transform of this rotation matrix:

$$\begin{bmatrix} n_i^I \\ n_j^I \\ n_k^I \end{bmatrix} = \begin{bmatrix} c\phi c\psi & s\theta s\phi c\psi - c\theta s\psi & c\theta s\phi c\psi + s\theta s\psi \\ c\phi s\psi & s\theta s\phi s\psi + c\theta c\psi & c\theta s\phi s\psi - s\theta c\psi \\ -s\phi & s\theta c\phi & c\theta c\phi \end{bmatrix} \begin{bmatrix} n_i^B \\ n_j^B \\ n_k^B \end{bmatrix} \tag{2.18}$$

Applying the rotation matrix from equation (2.17), we represent the gravitational force in

the body frame as:

$$\frac{f_G^B}{m} = \begin{bmatrix} -G\sin\phi \\ G\sin\theta\cos\phi \\ G\cos\theta\cos\phi \end{bmatrix} \tag{2.19}$$

Substituting the gravity and wing thrust forces into equation (2.13) we get:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} -G\sin\phi \\ G\sin\theta\cos\phi \\ G\cos\theta\cos\phi - \frac{T_1+T_2+T_3+T_4}{m} \end{bmatrix} - \begin{bmatrix} qw-rv \\ ru-pw \\ pv-qu \end{bmatrix} \tag{2.20}$$

In addition, we also require the translational velocities in the inertial frame $[\dot{x}, \dot{y}, \dot{z}]^T$, which we can find by applying the rotation matrix from equation (2.18) on the vector $[u, v, w]^T$:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} c\phi c\psi & s\theta s\phi c\psi - c\theta s\psi & c\theta s\phi c\psi + s\theta s\psi \\ c\phi s\psi & s\theta s\phi s\psi + c\theta c\psi & c\theta s\phi s\psi - s\theta c\psi \\ -s\phi & s\theta c\phi & c\theta c\phi \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{2.21}$$

Now that we have derived the translational kinematics in matrix form, we need to do the same for the rotational kinematics by representing equation (2.12) in matrix form. This requires knowing the moment of inertia $I$ which, for our three dimensional MAV, will be the symmetric three-by-three tensor:

$$I = \begin{bmatrix} I_{\theta\theta} & -I_{\theta\phi} & -I_{\theta\psi} \\ -I_{\phi\theta} & I_{\phi\phi} & -I_{\phi\psi} \\ -I_{\psi\theta} & -I_{\psi\phi} & I_{\psi\psi} \end{bmatrix} \tag{2.22}$$

Since the MAV is a rigid body with constant mass and we have defined its rotational axes

to lie along the principle moments of inertia, the tensor becomes the diagonal matrix:

$$I = \begin{bmatrix} I_\theta & 0 & 0 \\ 0 & I_\phi & 0 \\ 0 & 0 & I_\psi \end{bmatrix} \tag{2.23}$$

The torques acting on the vehicle in the body frame are $\tau^B = [L, M, N]^T$ and are shown in Figure 2.1. Representing equation (2.12) in matrix form, we get:

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} I_\theta & 0 & 0 \\ 0 & I_\phi & 0 \\ 0 & 0 & I_\psi \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} (I_\psi - I_\phi)qr \\ (I_\theta - I_\psi)pr \\ (I_\phi - I_\theta)pq \end{bmatrix} \tag{2.24}$$

In the body frame, torque is caused by thrust differentials about each of the three axes and is given by $\overrightarrow{\tau} = \overrightarrow{r} \times \overrightarrow{F}$. Since the wings are fixed to the body frame, the thrust vectors will always be perpendicular to each axis of rotation allowing our expression for torque to simplify to $\tau = rF$. For the MAV, $r$ is the distance from each wing to the rotation axis of concern. The MAV is symmetric about the three body frame principal planes so the four distances from each wing to the roll rotation axis are the same, to the pitch rotation axis are the same, and to the yaw rotation axis are the same. We represent these distances as $D_\theta$, $D_\phi$, and $D_\psi$. Using Figure 2.1 as a visual aid, we can express $[L, M, N]^T$ in terms of the wing thrusts and determine the sign of each thrust. Thrusts $T_2$ and $T_4$ produce positive torque, $L$, and $T_1$ and $T_3$ produce negative torque, $-L$. Thrusts $T_1$ and $T_2$ produce positive torque, $M$, and $T_3$ and $T_4$ produce negative torque, $-M$. Lastly, thrusts $T_2$ and $T_3$ produce positive torque, $N$, (due to the horizontal component of the thrusts) and $T_1$ and $T_4$ produce

negative torque, $-N$. Expressed in matrix form, we have:

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} D_\theta(-T_1+T_2-T_3+T_4) \\ D_\phi(T_1+T_2-T_3-T_4) \\ D_\psi(-T_1+T_2+T_3-T_4) \end{bmatrix} \tag{2.25}$$

Using equation (2.25), and the fact that $I^{-1} = \begin{bmatrix} I_\theta^{-1} & 0 & 0 \\ 0 & I_\phi^{-1} & 0 \\ 0 & 0 & I_\psi^{-1} \end{bmatrix}$,

we can rearrange equation (2.24) to get:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{D_\theta}{I_\theta}(-T_1+T_2-T_3+T_4) - \frac{1}{I_\theta}(I_\psi - I_\phi)qr \\ \frac{D_\phi}{I_\phi}(T_1+T_2-T_3-T_4) - \frac{1}{I_\phi}(I_\theta - I_\psi)pr \\ \frac{D_\psi}{I_\psi}(-T_1+T_2+T_3-T_4) - \frac{1}{I_\psi}(I_\phi - I_\theta)pq \end{bmatrix} \tag{2.26}$$

We will use this equation in our simulation model to find $[\dot{p}, \dot{q}, \dot{r}]^T$ and, through integration, $[p, q, r]^T$. Next, we must associate $[p, q, r]^T$ with the Euler angles $[\theta, \phi, \psi]^T$. We can express $\vec{\omega}$ in terms of two basis sets:

$$\vec{\omega} = p\,\vec{i}_B + q\,\vec{j}_B + r\,\vec{k}_B \tag{2.27}$$

$$\vec{\omega} = \dot{\theta}\,\vec{i}_B + \dot{\phi}\,\vec{j}_2 + \psi\,\vec{k}_1 \tag{2.28}$$

In equation (2.28), note that if we are to express the rotation of the body frame as seen by the inertial frame using Euler angles, then we have to use the intermediate reference frames that we passed through for each rotation in equations (2.14) through (2.16). We can find the transformation matrix between the two expressions of $\vec{\omega}$ by applying the rotation matrices from equations (2.14) to (2.16) to the basis vectors of $\vec{\omega}$ from equation (2.28). The rate of change in roll, $\dot{\theta}$, occurs about the body frame x-axis and thus needs no rotation. The rate

of change of pitch, $\dot{\phi}$, is about the 'frame 2' y-axis and thus requires a roll rotation, $R(\theta)$, to express it in the body frame. The rate of change of yaw, $\dot{\psi}$, is about the 'frame 1' z-axis and so needs a pitch rotation, $R(\phi)$, followed by a roll rotation, $R(\theta)$, to get it in the body frame. This transformation is expressed as:

$$\overrightarrow{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}_B = \begin{bmatrix} \dot{\theta} \\ 0 \\ 0 \end{bmatrix} + R(\theta) \begin{bmatrix} 0 \\ \dot{\phi} \\ 0 \end{bmatrix} + R(\theta)R(\phi) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \tag{2.29}$$

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix}_B = \overbrace{\begin{bmatrix} 1 & 0 & -\sin\phi \\ 0 & \cos\theta & \cos\phi\sin\theta \\ 0 & -\sin\theta & \cos\phi\cos\theta \end{bmatrix}}^{T} \begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix} \tag{2.30}$$

$T$ is the transformation matrix between the Euler angle expression of $\overrightarrow{\omega}$ and the body frame expression of $\overrightarrow{\omega}$. To get the rotation velocities in the inertial frame, we can use $[p, q, r]^T$ from equation (2.29) and the inverse of $T$ to calculate $[\dot{\theta}, \dot{\phi}, \dot{\psi}]^T$ as shown:

$$\begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix} = T^{-1} \begin{bmatrix} p \\ q \\ r \end{bmatrix}_B \tag{2.31}$$

where,

$$T^{-1} = \begin{bmatrix} 1 & \tan\phi\sin\theta & \tan\phi\cos\theta \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta/\cos\phi & \cos\theta/\cos\phi \end{bmatrix} \tag{2.32}$$

### 2.2.3 State Space Model

Our objective is to find $\dot{x} = f(x) + g(x)u$ in state space form. In order to describe the MAV's state, the state vector must include at least twice as many elements as number of degrees of freedom in our system. Since the reference signals will be given in the inertial frame basis set, and the kinematics will be calculated in the body frame, our state vector will need to include the state of the MAV in both reference frames. Our MAV possesses six degrees of freedom, three rotational and three translational, so our state vector, $X$, must possess twelve elements: three inertial translational, three body frame translational, three inertial rotational, and three body frame rotational as shown below:

$$\dot{X} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \dot{u} & \dot{v} & \dot{w} & \dot{\theta} & \dot{\phi} & \dot{\psi} & \dot{p} & \dot{q} & \dot{r} \end{bmatrix}^T \tag{2.33}$$

Using equations (2.20), (2.21), (2.26), and (2.31), in equation (2.33), we get:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} (c\phi c\psi)u + (s\theta s\phi c\psi - c\theta s\psi)v + (c\theta s\phi c\psi + s\theta s\psi)w \\ (c\phi s\psi)u + (s\theta s\phi s\psi + c\theta c\psi)v + (c\theta s\phi s\psi - s\theta c\psi)w \\ (-\sin\phi)u + (\sin\theta\cos\phi)v + (\cos\theta\cos\phi)w \\ -G\sin\phi - qw + rv \\ G\sin\theta\cos\phi - ru + pw \\ G\cos\theta\cos\phi - \frac{T_1+T_2+T_3+T_4}{m} - pv + qu \\ p + \tan\phi\sin\theta\, q + \tan\phi\cos\theta\, r \\ \cos\theta\, q - \sin\theta\, r \\ \frac{\sin\theta}{\cos\phi}q + \frac{\cos\theta}{\cos\phi}r \\ \frac{D_\theta}{I_\theta}(-T_1 + T_2 - T_3 + T_4) - \frac{1}{I_\theta}(I_\psi - I_\phi)qr \\ \frac{D_\phi}{I_\phi}(T_1 + T_2 - T_3 - T_4) - \frac{1}{I_\phi}(I_\theta - I_\psi)pr \\ \frac{D_\psi}{I_\psi}(-T_1 + T_2 + T_3 - T_4) - \frac{1}{I_\psi}(I_\phi - I_\theta)pq \end{bmatrix} \tag{2.34}$$

Since there are four actuators in the system, our input vector, $u$, will consist of each of the four thrusts provided by the wings, $[T_1, T_2, T_3, T_4]^T$. Separating the input out of the matrix in equation (2.34), we can find the matrices $f(x)$ and $g(x)$ in the desired form $\dot{x} = f(x) + g(x)u$.

$$
\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \overbrace{\begin{bmatrix} (c\phi c\psi)u + (s\theta s\phi c\psi - c\theta s\psi)v + (c\theta s\phi c\psi + s\theta s\psi)w \\ (c\phi s\psi)u + (s\theta s\phi s\psi + c\theta c\psi)v + (c\theta s\phi s\psi - s\theta c\psi)w \\ (-\sin\phi)u + (\sin\theta\cos\phi)v + (\cos\theta\cos\phi)w \\ -G\sin\phi - qw + rv \\ G\sin\theta\cos\phi - ru + pw \\ G\cos\theta\cos\phi - pv + qu \\ p + \tan\phi\sin\theta\, q + \tan\phi\cos\theta\, r \\ \cos\theta\, q - \sin\theta\, r \\ \frac{\sin\theta}{\cos\phi}q + \frac{\cos\theta}{\cos\phi}r \\ -\frac{1}{I_\theta}(I_\psi - I_\phi)qr \\ -\frac{1}{I_\phi}(I_\theta - I_\psi)pr \\ -\frac{1}{I_\psi}(I_\phi - I_\theta)pq \end{bmatrix}}^{f(x)}
$$

$$
+ \overbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{1}{m} & -\frac{1}{m} & -\frac{1}{m} & -\frac{1}{m} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{D_\theta}{I_\theta} & \frac{D_\theta}{I_\theta} & -\frac{D_\theta}{I_\theta} & \frac{D_\theta}{I_\theta} \\ \frac{D_\phi}{I_\phi} & \frac{D_\phi}{I_\phi} & -\frac{D_\phi}{I_\phi} & -\frac{D_\phi}{I_\phi} \\ -\frac{D_\psi}{I_\psi} & \frac{D_\psi}{I_\psi} & \frac{D_\psi}{I_\psi} & -\frac{D_\psi}{I_\psi} \end{bmatrix}}^{g(x)} \overbrace{\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix}}^{u} \tag{2.35}
$$

# CHAPTER III

# NONLINEAR CONTROL SYSTEM

## 3.1 Controller Derivation

### 3.1.1 Controller Setup

Now that we have a state-space model for the MAV, we need to derive a controller which will provide set point and tracking control for the desired state variables. In our simulation, as in our hardware implementation, we are primarily concerned with controlling the MAV's attitude tracking capabilities. In our hardware test bench, the MAV will be fixed to a platform that constrains it to three degrees of rotational freedom but, in our simulation, we will also control altitude holding for hovering tests. So, for our simulation, the output $h(x)$ we want to control is $[\theta, \phi, \psi, z]^T$. We then require our state vector to include the inertial rotational velocities $[\dot{\theta}, \dot{\phi}, \dot{\psi}]^T$, the body frame rotational velocities $[\dot{p}, \dot{q}, \dot{r}]^T$, and the altitude velocity and acceleration $[\dot{z}, \ddot{z}]^T$. The portion of the model that we will be controlling is:

$$
\begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \\ \dot{p} \\ \dot{q} \\ \dot{r} \\ \dot{z} \\ \ddot{z} \end{bmatrix} = \overbrace{\begin{bmatrix} p + \tan\phi \sin\theta\, q + \tan\phi \cos\theta\, r \\ \cos\theta\, q - \sin\theta\, r \\ \frac{\sin\theta}{\cos\phi} q + \frac{\cos\theta}{\cos\phi} r \\ -\frac{1}{I_\theta}(I_\psi - I_\phi) qr \\ -\frac{1}{I_\phi}(I_\theta - I_\psi) pr \\ -\frac{1}{I_\psi}(I_\phi - I_\theta) pq \\ \int \ddot{z} dt \\ G + \cos\theta \cos\phi(-pv + qu) \end{bmatrix}}^{f(x)} + \overbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{D_\theta}{I_\theta} & \frac{D_\theta}{I_\theta} & -\frac{D_\theta}{I_\theta} & \frac{D_\theta}{I_\theta} \\ \frac{D_\phi}{I_\phi} & \frac{D_\phi}{I_\phi} & -\frac{D_\phi}{I_\phi} & -\frac{D_\phi}{I_\phi} \\ -\frac{D_\psi}{I_\psi} & \frac{D_\psi}{I_\psi} & \frac{D_\psi}{I_\psi} & -\frac{D_\psi}{I_\psi} \\ 0 & 0 & 0 & 0 \\ -\frac{\cos\theta\cos\phi}{m} & -\frac{\cos\theta\cos\phi}{m} & -\frac{\cos\theta\cos\phi}{m} & -\frac{\cos\theta\cos\phi}{m} \end{bmatrix}}^{g(x)} \overbrace{\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix}}^{u} \tag{3.1}
$$

The input to the system is the wing thrusts $T_1$, $T_2$, $T_3$, $T_4$. The system contains eight states, four outputs, and four inputs, which is appropriate since a Multiple Input Multiple Output (MIMO) system requires an equal number of inputs and outputs [4].

A common approach to controlling a nonlinear system such as our MAV model is to linearize it and then to apply a linear controller to the linearized system. There are several methods of linearization. One method is to linearize the system offline using Jacobian linearization. However, this only renders an exact representation of the nonlinear model at

the stabilization point and deviations from that point create instability. On the contrary, using online feedback linearization renders an exact representation of the nonlinear system across a large area of set points providing a wider stability range. The most common methods of feedback linearization are input-state linearization and input-output linearization. Input-state linearization attempts to linearize a map between transformed state variables and transformed inputs and to create a set of outputs which realize this linearization. However, this method can often complicate controller design when the mapping between the transformed inputs and actual outputs is nonlinear. We will choose input-output linearization which linearizes a mapping between transformed inputs, $v$, and the actual outputs, $y$. A linear controller can then be applied to the resulting linear mapping. Input-output linearization uses two loops as shown in Figure 3.1.

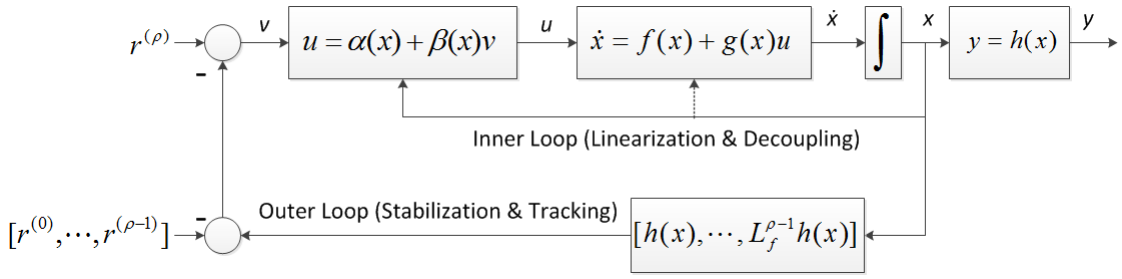



Figure 3.1: Input-Output Linearization Diagram

The inner loop performs linearization and decoupling using a nonlinear mapping from the transformed input $v$ to the actual input $u$ in the form of the control law:

$$u = \alpha(x) + \beta(x)v \tag{3.2}$$

The matrix function $\alpha(x)$ and $\beta(x)$ should be chosen to form a linear input-output map between $v$ and $y$. The outer loop is the linear controller which will produce stable set point and tracking behavior.

### 3.1.2 Stability Analysis

In order to apply input-output linearization on this system, we must verify that the internal system dynamics are bounded by determining if the system is minimum-phase. To this end we can test the system's zero dynamics for asymptotic stability. This is important in a system since even if the output is at an equilibrium, the internal dynamics may not be at equilibrium and may, in fact, be unstable. We can expose these internal dynamics (zero dynamics) by forcing the output to equilibrium. In general, the dimension of the zero dynamics is equal to the difference between the number of state variables in the system, $n$, and the relative degree of the system, $\rho$.

$$\dim(\textit{zero dynamics}) = n - \rho \tag{3.3}$$

If the dimension of the zero dynamics is zero, then we have no internal dynamics and the system is trivially minimum phase. For our system, we have eight state variables $[\dot{\theta}, \dot{\phi}, \dot{\psi}, \dot{p}, \dot{q}, \dot{r}, \dot{z}, \ddot{z}]^T$ and we must determine the relative degree.

Relative degree is inherent to the system and is found by determining the number of times we must differentiate each of the outputs with respect to time before an invertible dependence on the input appears. As shown below, the second derivative of each output $y_j$ depends on the input since $\dot{p}$, $\dot{q}$, $\dot{r}$, and $\dot{w}$ depend explicitly on $u$.

$$\begin{aligned}
\ddot{y}_1 &= \dot{p} + \dot{\theta} q \cos\theta \tan\phi + \dot{\phi} q \sin\theta \sec^2\phi + \dot{q} \sin\theta \tan\phi \\
&\quad -\dot{\theta} r \sin\theta \tan\phi + \dot{\phi} r \cos\theta \sec^2\phi + \dot{r} \cos\theta \tan\phi \\
\ddot{y}_2 &= -\dot{\theta} q \sin\theta + \dot{q} \cos\theta - \dot{\theta} r \cos\theta - \dot{r} \sin\theta \\
\ddot{y}_3 &= \dot{\theta} q \frac{\cos\theta}{\cos\phi} - \dot{\phi} q \frac{\sin\theta}{\cos^2\phi} + \dot{q} \frac{\sin\theta}{\cos\phi} - \dot{\theta} r \frac{\sin\theta}{\cos\phi} - \dot{\phi} r \frac{\cos\theta}{\cos^2\phi} + \dot{r} \frac{\cos\theta}{\cos\phi} \\
\ddot{y}_4 &= G + \cos\theta \cos\phi \left(-pv + qu - \frac{T_1 + T_2 + T_3 + T_4}{m}\right)
\end{aligned} \tag{3.4}$$

Thus, $\rho_j = 2$ for each output $y_j$, the total relative degree $\rho_t = \rho_1 + \rho_2 + \rho_3 + \rho_4$ is equal to eight. Since both the number of state variables, $n$, and the total relative degree of the system, $\rho_t$, are eight, the dimension of the zero dynamics is zero. Thus, we have no unobserved internal dynamics and our system is trivially minimum phase. Note that it is also required that the dependence of $y$ on $u$ be invertible. This will be shown in the next section.

### 3.1.3 Input-Output Linearization

We desire to find an input feedback law that will linearize the mapping from our output $y$ to our transformed input $v$. Since we know that the second derivative of our output depends explicitly on $u$ which depends on $v$, let us investigate the second derivative of $y$. The derivative, $\dot{y}$, can be shown to be:

$$\dot{y} = \frac{\partial h}{\partial x}[f(x) + g(x)u] \tag{3.5}$$

This can be simplified using lie derivative notation, defined as:

$$L_f h(x) = \frac{\partial h(x)}{\partial x} f(x) \tag{3.6}$$

Using this notation, derivatives of $y$ become:

$$\begin{aligned} \dot{y} &= L_f h(x) + L_g h(x) u \\ \ddot{y} &= L_f^2 h(x) + L_g L_f h(x) u \\ &\vdots \end{aligned} \tag{3.7}$$

or, in general:

$$y^{(\rho)} = L_f^{\rho} h(x) + L_g L_f^{\rho-1} h(x) u \tag{3.8}$$

Representing equation (3.8) in matrix form for our relative degree, $\rho = 2$, we get:

$$\begin{bmatrix} \ddot{y_1} \\ \ddot{y_2} \\ \ddot{y_3} \\ \ddot{y_4} \end{bmatrix} = \overbrace{\begin{bmatrix} L_f^2 h_1(x) \\ L_f^2 h_2(x) \\ L_f^2 h_3(x) \\ L_f^2 h_4(x) \end{bmatrix}}^{B(x)} + \overbrace{\begin{bmatrix} L_{g_1} L_f h_1(x) & \cdots & L_{g_4} L_f h_1(x) \\ \vdots & \ddots & \vdots \\ L_{g_1} L_f h_4(x) & \dots & L_{g_4} L_f h_4(x) \end{bmatrix}}^{A(x)} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \tag{3.9}$$

Matrices $A(x)$ and $B(x)$ written explicitly in terms of state variables can be found in Appendix A. Solving equation (3.9) for $U$ gives:

$$U = A^{-1}(x)(\ddot{Y} - B(x)) \tag{3.10}$$

Inspecting equations (3.2) and (3.10), we can linearize the relationship between output $y$ and our transformed input $v$ by choosing $\alpha(x) = -A^{-1}(x)B(x)$ and $\beta(x) = A^{-1}(x)$. This choice cancels the nonlinear terms giving:

$$y^{(\rho)} = v \tag{3.11}$$

This mapping is a series of $\rho$ integrators of $v$. Since we are dividing by $L_g L_f^{\rho-1} h(x)$, it is required that it be invertible. To confirm invertibility, we must show that $\det A(x) \neq 0$. This determinant can be found (Appendix A) to be:

$$\det \begin{bmatrix} L_{g_1} L_f h_1(x) & \cdots & L_{g_4} L_f h_1(x) \\ \vdots & \ddots & \vdots \\ L_{g_1} L_f h_4(x) & \dots & L_{g_4} L_f h_4(x) \end{bmatrix} = \frac{16 D^3 \cos\theta}{I_\psi I_\phi I_\theta m} \tag{3.12}$$

Thus, $L_g L_f^{\rho-1} h(x)$ is invertible so long as $\theta$, the vehicle's roll, is not equal to $\frac{\pi}{2}$.

### 3.1.4 Outer-Loop Linear Controller

Now that we have linearized the relationship between our output and our input $v$, we must implement a linear controller which will cause each $y_j$ to track the reference values, $y_j^d$, that is $y \to y_j^d$ as $t \to \infty$. We define the set point error to be $e_j = y_j - y_j^d$ and velocity and acceleration errors to be $\dot{e}_j = \dot{y}_j - \dot{y}_j^d$ and $\ddot{e}_j = \ddot{y}_j - \ddot{y}_j^d$, respectively. For $y_j$, $\dot{y}_j$, and $\ddot{y}_j$ to approach $y_j^d$, $\dot{y}_j^d$, and $\ddot{y}_j^d$ as $t \to \infty$, the errors $e_j$, $\dot{e}_j$, and $\ddot{e}_j$ must satisfy the differential equation:

$$\ddot{e}_j + k_d \dot{e}_j + k_p e_j = 0 \tag{3.13}$$

Substituting the output variables for the errors and using equation (3.11), we get:

$$v_i = -k_p(y_j - y_j^d) - k_d(\dot{y}_j - \dot{y}_j^d) + \ddot{y}_j^d \tag{3.14}$$

Using this in our control law, equation (3.2), with our choices for $\alpha(x)$ and $\beta(x)$, we get the input to be:

$$U = A^{-1}(x)\,V - A^{-1}(x)\,B(x) \tag{3.15}$$

This equation written in terms of state variables and reference signals can be found in Appendix A.

## 3.2 Simulation

The purpose of the simulation is to demonstrate that a feedback linearization based controller can be used to control the three rotational degrees of freedom of a MAV with four actuators creating thrusts according to the flight mechanics defined in Section 1.3. The simulation was implemented using MATLAB Simulink.

The simulation constants, including vehicle dimensions, are initialized in 'MAV-Initialize.m' (Appendix B). The simulation model is implemented in 'simulation.mdl' which is shown in Appendix B. Each iteration of the model defines reference values for roll, pitch, and yaw

angles, velocities, and accelerations. Using these reference values, the nonlinear controller calculates the input vector using Equation (3.15) (Appendix A). Each element of the input vector is saturated at a maximum of one Newton of thrust, assuming a lifting capability of one hundred grams per wing, and a minimum thrust of zero, since the wings can only thrust downward (positive thrust). The input vector is sent to an s-function, defined in 'MAV-nonlinear-Dynamics.m', which calculates the current time step's state vector using the derivative of the state vector found from the nonlinear state-space model in Equation (2.34) implemented in 'MAV-nonlinear-model.m'. This state vector is plotted to show the orientation of the vehicle as it responds to the reference signals. Figure 3.2 shows the step response of the vehicle's roll, pitch, yaw, and z when the step function was applied to the roll, pitch, yaw, and z reference values simultaneously.

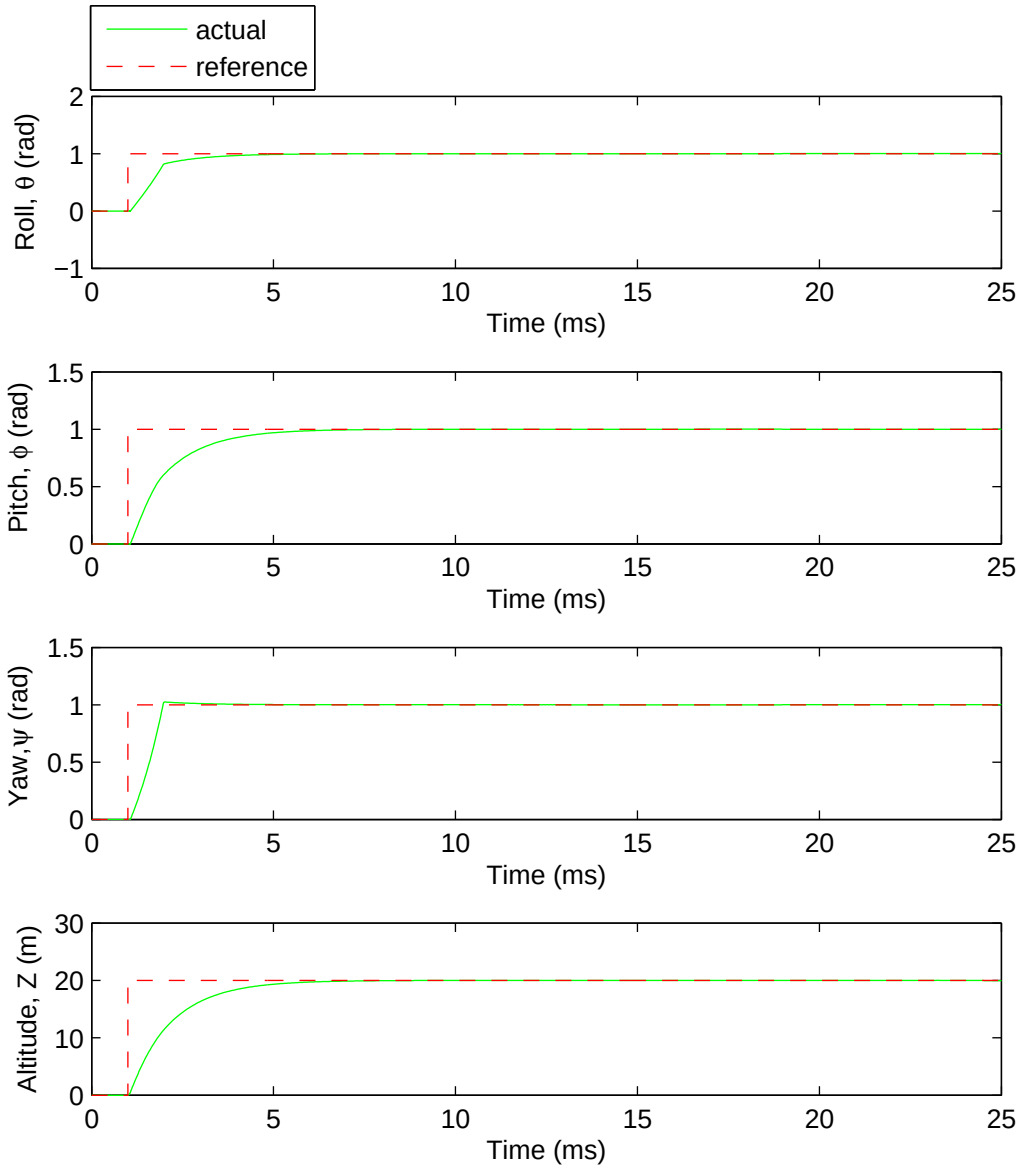



Figure 3.2: Step Response: Roll, Pitch, Yaw, Z

Since the outer-loop linear controller provides tracking capabilities using derivatives of the error, Figure 3.3 shows the vehicle tracking a sinusoidal roll, pitch, and yaw reference signal of increasing frequency from 0.01 Hz to 0.5 Hz with amplitude 1 radians in 25 seconds as shown. Figure 3.4 shows the input thrusts calculated to produce such tracking.

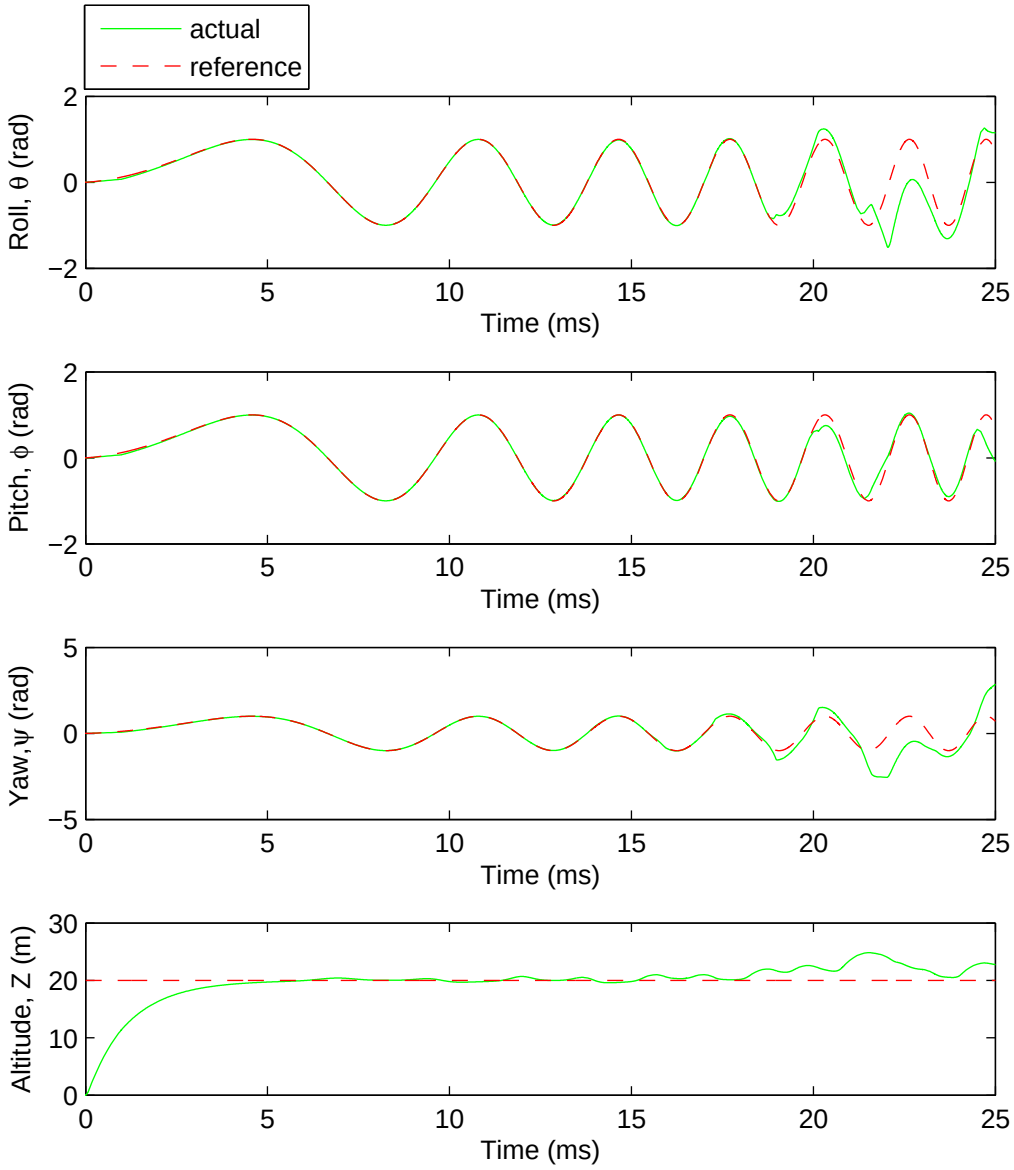



Figure 3.3: Tracking Response: Roll, Pitch, Yaw, Z

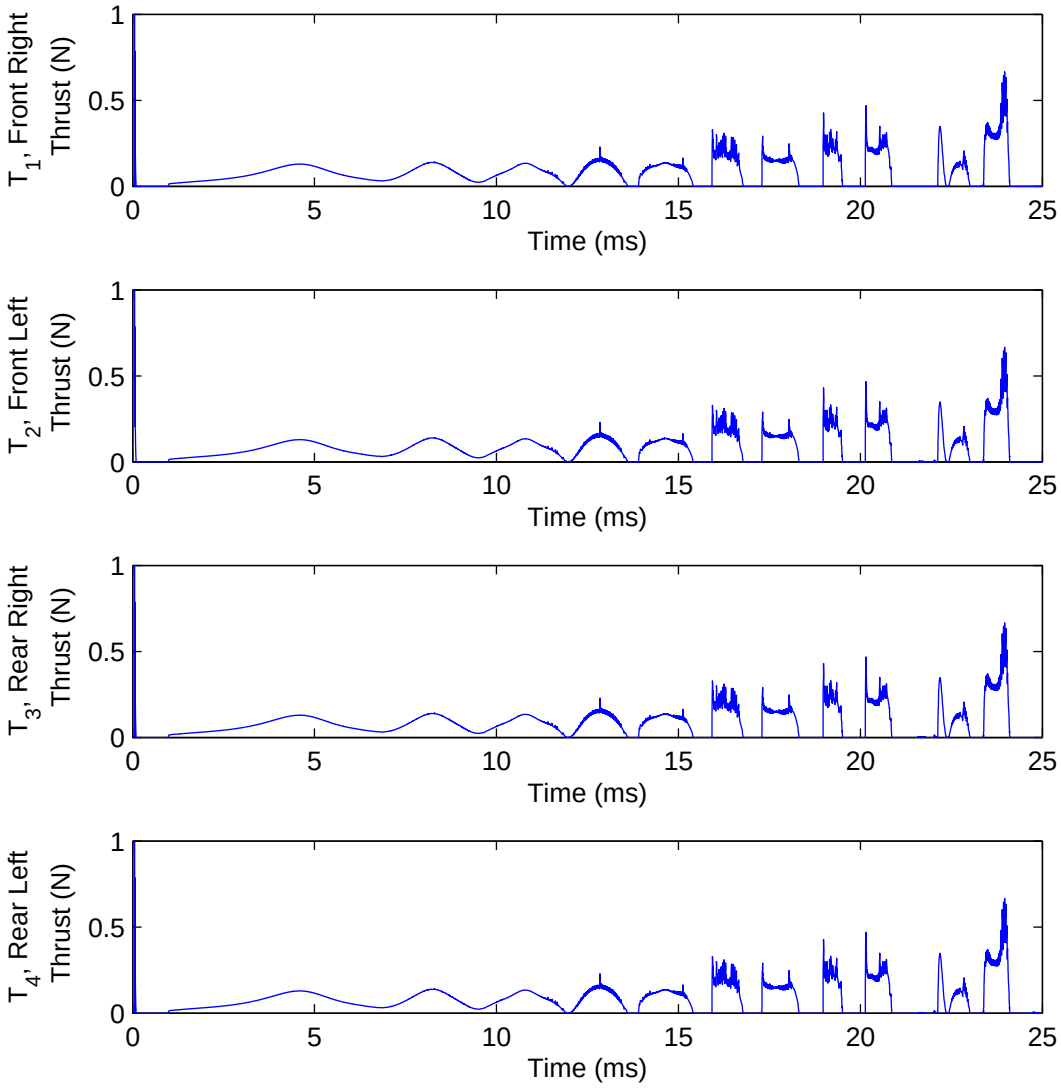



Figure 3.4: Wing Thrusts Used for Tracking Response

While the sinusoid frequency is less than ~0.4 Hz, the vehicle is capable of tracking the trajectory. However, as the frequency approaches 0.5 Hz, the inability of the vehicle to output negative thrusts decreases its agility and it begins to deviate from the reference trajectory. The input thrusts values differ for each motor in the hundredths place which is not resolvable on these graphs.

# CHAPTER IV

# LINEAR CONTROL AND HARDWARE IMPLEMENTATION

## 4.1 Overview

This chapter delineates the test bench design and implementation for the MAV concept. The purpose of the test bench is to verify in hardware that we can produce the desired pitch, roll, and yaw motions based on the flight mechanics we defined in Section 1.3. In choosing a control algorithm, we are limited by the computational power of the test bench processor. The input vector from the feedback linearization controller derived in chapter three requires a prohibitively large number of multiplications and additions for execution on the test bench microprocessor. In addition, the highly dynamic nature of the test bench requires a low control loop latency for controlling its motion. Thus, the linear-proportional (PD) controller described in this chapter is more suitable for testing the response of the test bench to reference step inputs.

## 4.2 Three Degree of Freedom Test Bench Design

### 4.2.1 Mechanical Structure

The test bench chassis uses carbon fiber rods constructed in an 'I' shape (Figure 4.1) with the twelve inch center shaft securing the avionics board. The avionics board, with gyroscopes and accerometers, is positioned at the center of gravity of the test bench so that linear acceleration and rotational rate measurements are taken about the center of mass. Each 5.75 inch crossbar in front and back supports a wing/actuator system at each tip. Each wing, with a height of 3.75 inches, is oriented vertically for maximum lift with a slight angle to provide horizontal thrust for acheiving yawing motion. The structure is mounted on a ball joint allowing for rotational freedom in pitch, roll, and yaw.

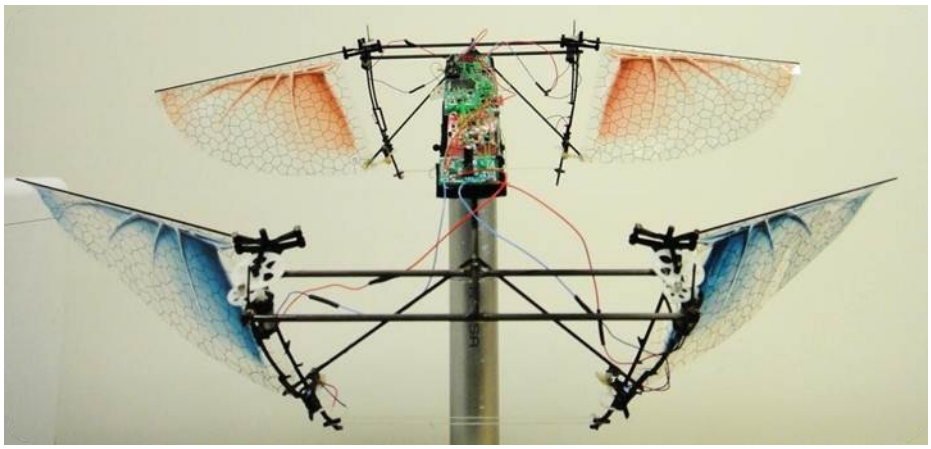


Figure 4.1: Test Bench

### 4.2.2 Electronics

The avionics board features a dsPIC33FJ256GP710A Microchip microcontroller which interfaces with sensors, performs controls computations, and communicates with the ground station. The avionics board electronics include three-axis gyroscopes, three-axis accelerometers, a quad motor controller, and a wireless transceiver as shown in Figure 4.2. The electronics are powered by a single-cell 3.3V Lithium-Polymer battery.

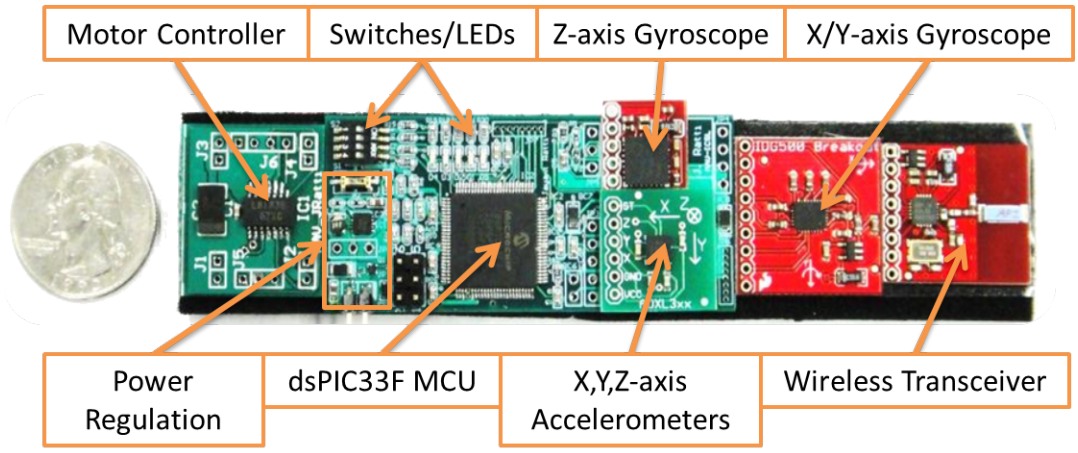



Figure 4.2: Avionics Board

The gyroscopes (IDG-500) and accelerometers (ADXL3XX) output analog voltages in proportion to the vehicle's rotational rate and linear accelerations respectively. The dsPIC

samples these analog signals at 12-bit resolution and stores them in memory using Direct Memory Access to avoid consuming CPU resources. The dsPIC33 processes this data using a low-pass filter to reduce the flapping-wing vibrations in the sensor readings.

The nRF24L01+ is a 2.4GHz transceiver that features automatic packet handling (attaching headers to each data packet) and automatic transaction handling (tracking acknowledgements and resending dropped packets). The dsPIC relays data to and configures the transceiver via SPI protocol. The transceiver then transmits this data to the ground station.

The motor controller receives PWM signals from the dsPIC33 for each motor and outputs a corresponding analog voltage to each motor. The dsPIC33 features a PWM module providing automated PWM signal generation based on period and duty cycle configurations set in software.

## 4.3 Controller Algorithm Implementation

The test bench algorithm, depicted as a flowchart in Figure 4.3, is logically divided into an initialization section and the cyclic control loop. The code can be found in Appendix C.

### 4.3.1 Initialization Section

The initialization section performs three tasks: setting up variables/memory required by the algorithm, configuring hardware peripherals, and calibrating the sensors. At startup the first part of the initialization section configures the dsPIC's PLL's to increase the external 8MHz oscillator frequency to 25MHz in order to reduce the control loop latency. It also allocates 28KB of RAM to store flight logs for transmission to the ground station at the completion of each test.

The second part initializes five peripherals on the dsPIC as well as our transceiver module. The first of these peripherals is the SPI module which is required to communicate with the transceiver.

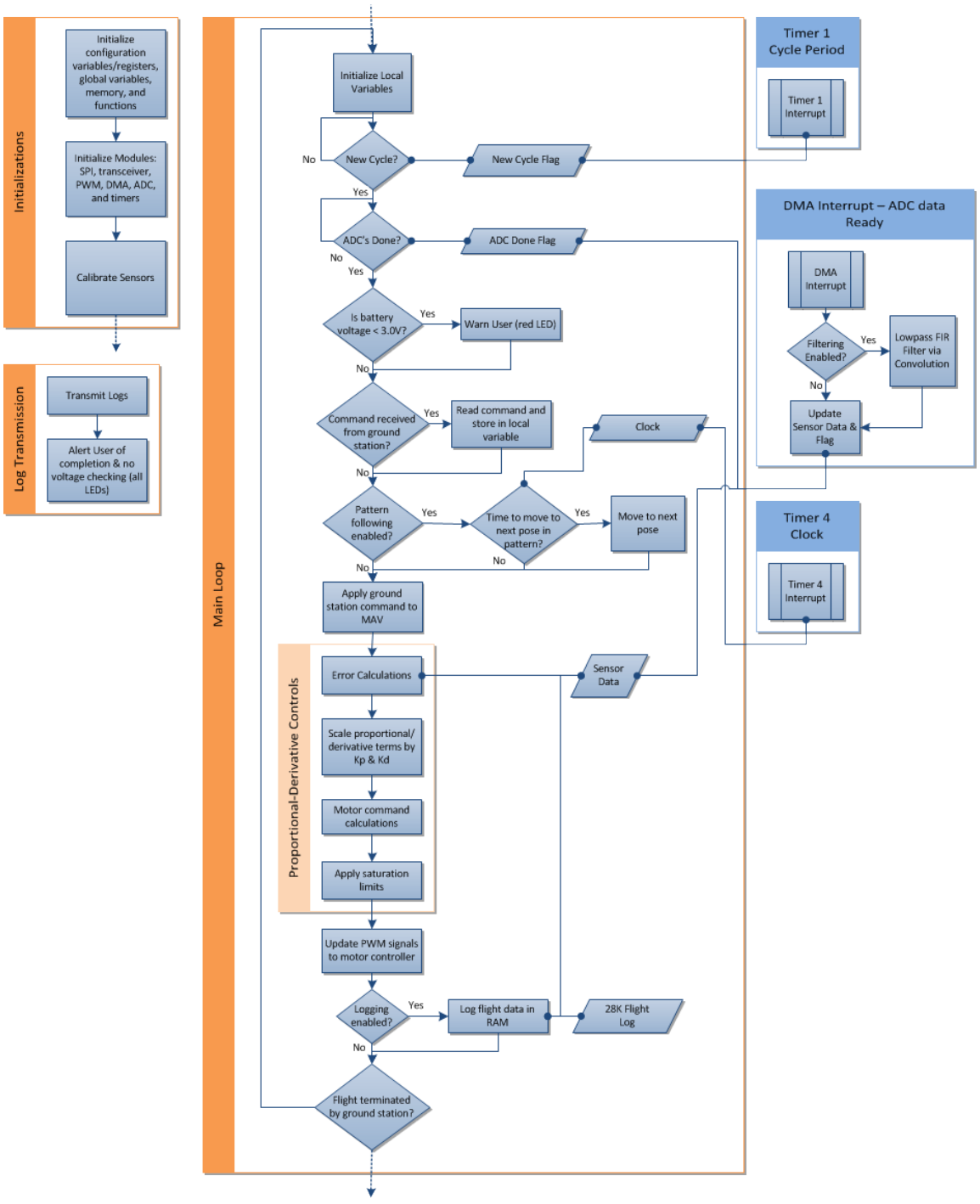



Figure 4.3: Algorithm Flowchart

Using SPI, we configure the transceiver chip to perform a cyclic redundancy check (CRC), to automatically send acknowledgements as well as automatic retransmit packets up to 15 times in the event of not receiving an acknowledgement within 500μs of transmission.

Thirdly, to read the analog gyroscope and accelerometer sensors, we configure the analog-to-digital converter (ADC) module to sample the signals at 12-bit resolution in channel scanning mode across nine channels. The first three selected channels are connected to the accelerometers, the next three to the gyroscopes, and the last three to the battery, power, and VDD voltages.

To prevent the ADC's from consuming valuable CPU cycles, we initialize the Direct Memory Access module to allow the ADC to write to RAM directly. The DMA is configured to thrown an interrupt after every nine transfers so the CPU can process the data.

To control each motor, the dsPIC uses its output compare registers in PWM mode to send PWM signals to the motor controller which outputs corresponding analog voltages to each motor.

Lastly, two timers are initialized for use in control loop timing management and maintaining a second/milliseconds clock which is used for pattern following in the control loop as well as time-stamping the flight logs. The period of the first timer dictates the control loop frequency which, in our case, reaches a maximum of 71 Hz. The period of the second timer is set to trigger the timer's interrupt service routine to update the clock every millisecond.

The third part of the initialization section performs sensor calibration using an averaging function. This calibration is necessary to get an accurate estimate of the horizontal/level orientation of the test bench. While the calibration function is running, the test bench must remain level. The algorithm samples the nine channels (accels, gyros, and voltage levels) fifty times and averages the data sample collection for each channel and sets the results to be the initial reference values.

### 4.3.2 Filter Implementation

After all the sensors are sampled by the ADC, the algorithm filters the data using a low-pass FIR filter of order eleven and even symmetry with the stop band starting at 6 Hz. The FIR filter convolution is performed in two steps: shifting and multiply-accumulate. The shifting is implemented using circular buffers (one for each of the nine channels) which reduce latency by moving pointers rather than memory. As integer multiplication is significantly faster than floating-point multiplication on the dsPIC, the taps are scaled and truncated to allow the multiply-accumulate process to call only integer operations. Figure 4.4 shows a test in which gyroscope and accelerometer measurement were unfiltered and Figure 4.5 shows an identical test in which said measurements were filtered.

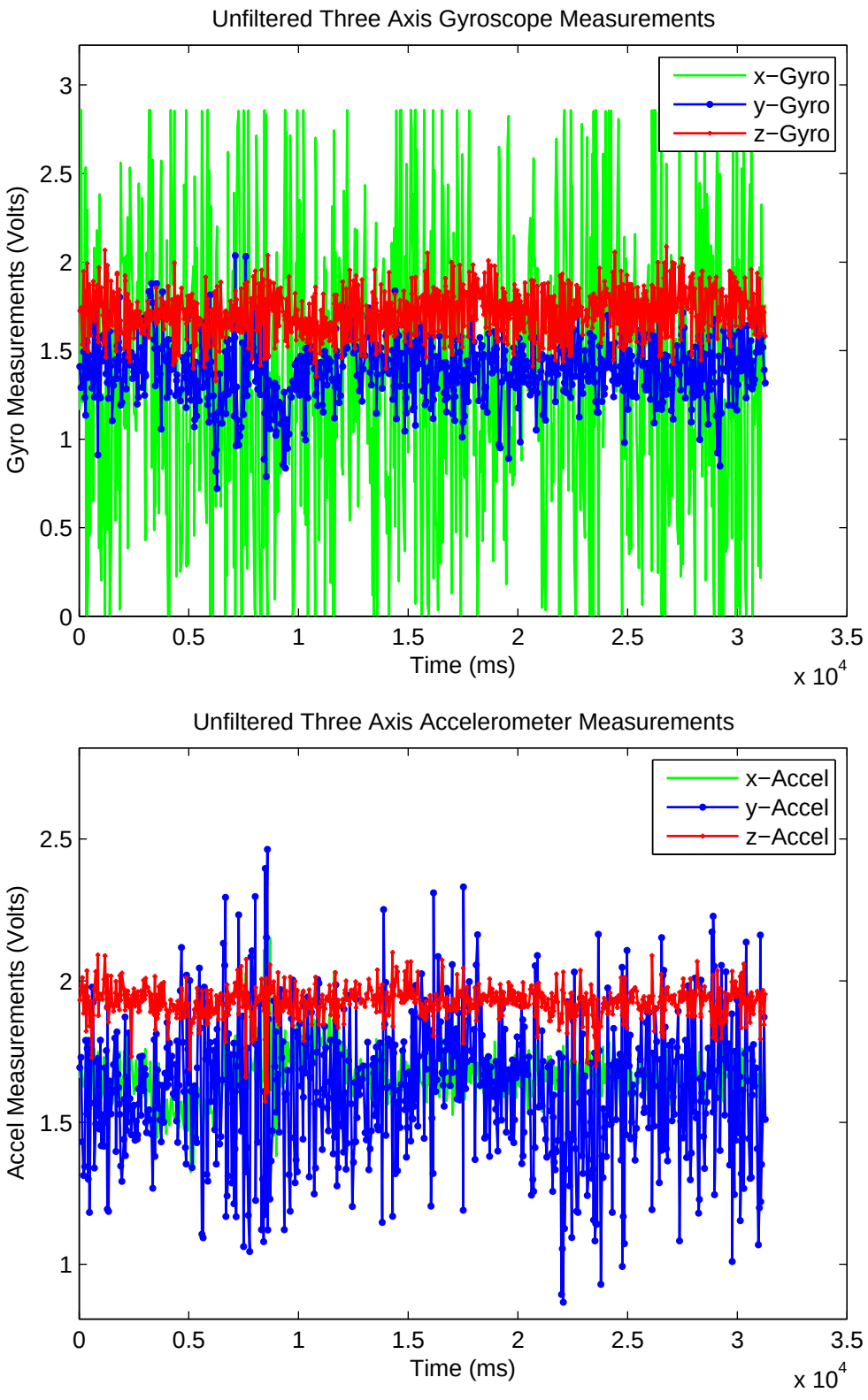



Figure 4.4: Unfiltered Gyroscope and Accelerometer Measurements

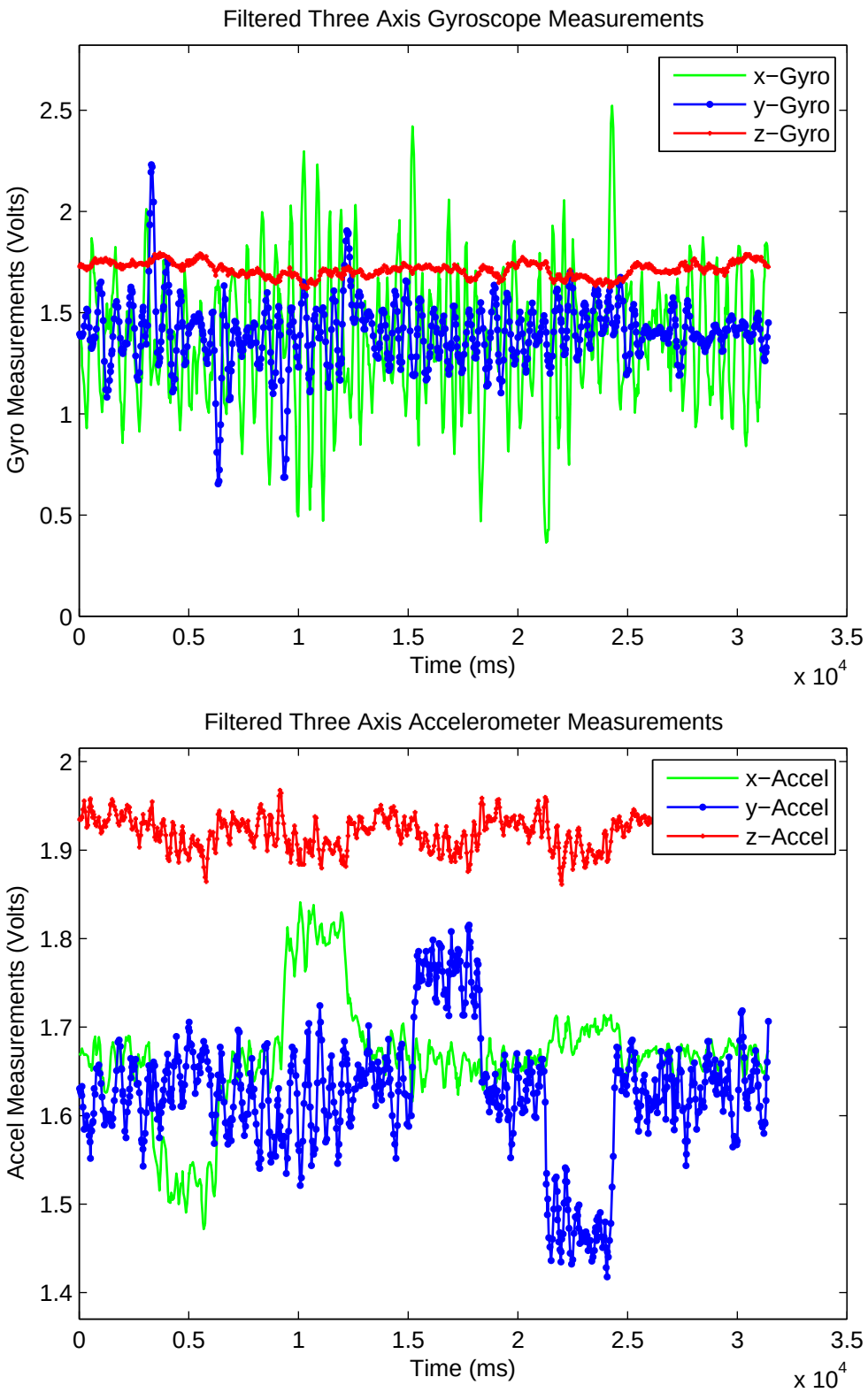



Figure 4.5: Filtered Gyroscope and Accelerometer Measurements

### 4.3.3 Control Algorithm

On each cycle, the algorithm checks the battery voltage to ensure the hardware is not damaged by low voltage and a led alert is triggered if required. We then poll the transceiver for received data from the ground station and set relevant flags if a command is received (each command's affect can be viewed in Figure 4.8). If pattern following is enabled and the current pose has expired, the test bench proceeds to the next pose. At this point, sensor measurements and reference values are ready for use in the PD control loop calculations.

We apply PD control as shown in Equation (4.1) to each of the three rotations, roll, pitch, and yaw. The rotation error $e$, as defined in 3.1.4, is the difference between the reference and measured values for rotation in degrees and its derivative is the rate of rotation, degrees per second. The proportional gain, $k_p$, is given in Newtons per degree and the derivative gain, $k_d$, is given in Newtons per degree per second. The input variable, $U$, defined as in section 3.1.4, is in Newtons.

$$U = K_p e + K_d \frac{d}{dt} e \tag{4.1}$$

Although accelerometers measure linear acceleration, we can use the accelerometers to measure rotation by measuring components of gravitational acceleration along each axis. For example, if the test bench pitches, the x-axis accelerometer will measure an acceleration, $a_{accel}$, due to the component of gravitational acceleration along the x-axis as shown in Figure 4.6.

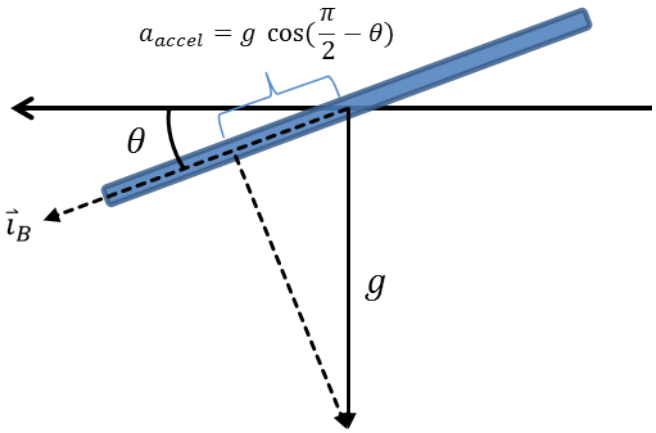



Figure 4.6: Measuring rotation using accelerometers

The angle of rotation can be found using Equation (4.2).

$$\theta = \frac{\pi}{2} - \cos^{-1}(\frac{a_{accel}}{g}) \tag{4.2}$$

Since a roll rotation introduces a component of gravitational acceleration along the y-axis, the y-axis accelerometer measures roll rotation. Thus, we can find the rotation error, $e$, as the difference between the reference angle and accelerometer measurement given in degrees from Equation (4.2). Note that the accelerometers cannot detect a yaw rotation when the vehicle remains level since the direction of gravity remains constant relative to the accelerometers. Since the gyroscopes measure rotational velocity, the rotation rate error, $\frac{d}{dt}e$, is the difference between the reference rotational velocity (always zero in our case) and the measured rotational velocity given in degrees per second. The rotational error for pitch, roll, and yaw is represented by $e_{pitch}$, $e_{roll}$, and $e_{yaw}$ respectively and their corresponding gains are represented by $k_p^{pitch}$, $k_p^{roll}$, and $k_p^{yaw}$, respectively. Similarly rotational velocity errors for pitch, roll, and yaw are represented by $e_{pitchrate}$, $e_{rollrate}$, and $e_{yawrate}$ respectively and their corresponding gains are represented by $k_d^{pitch}$, $k_d^{roll}$, and $k_d^{yaw}$, respectively. Thus, for each rotation axis (pitch, roll, and yaw) we calculate the control input by multiplying the accelerometer error by the proportional gain and the gyroscope error by the derivative

gain as shown in equation (4.3).

$$\begin{aligned} U_{pitch} &= k_p^{pitch} e_{pitch} + k_d^{pitch} e_{pitchrate} \\ U_{roll} &= k_p^{roll} e_{roll} + k_d^{roll} e_{rollrate} \\ U_{yaw} &= k_p^{yaw} e_{yaw} + k_d^{yaw} e_{yawrate} \end{aligned} \tag{4.3}$$

The PD gains are found using the Ziegler–Nichols tuning method and are listed in Table 4.1

Table 4.1: PD Gains

| | Kp | Kd |
|---|---|---|
| Pitch | 2.05 | 0.5 |
| Roll | 0.85 | 0.1 |
| Yaw | 1 | 0.35 |

If logging is enabled, the algorithm stores the current iteration's flight data in the flight logs on RAM. When the ground station terminates the test, the algorithm transmits the 28KB flight logs to the ground station for analysis.

## 4.4 Ground Station

### 4.4.1 Overview

The function of the ground station is to relay commands from the user to the test bench and to receive and display flight logs from the test bench after each test. The ground station is divided into transmission and reception sections which each have a dedicated transceiver for transmitting or receiving data as shown in Figure 4.7.

Transmitting commands to the test bench is handled by DSA's MTTTY program which relays characters entered on the keyboard to the transmitting transceiver via serial. Figure 4.8 shows the relevant keyboard commands.

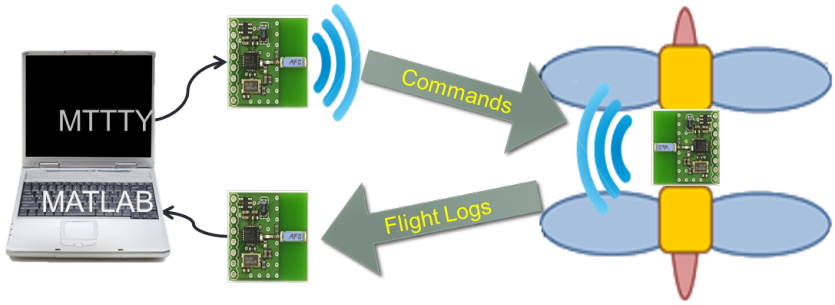



Figure 4.7: Wireless Setup

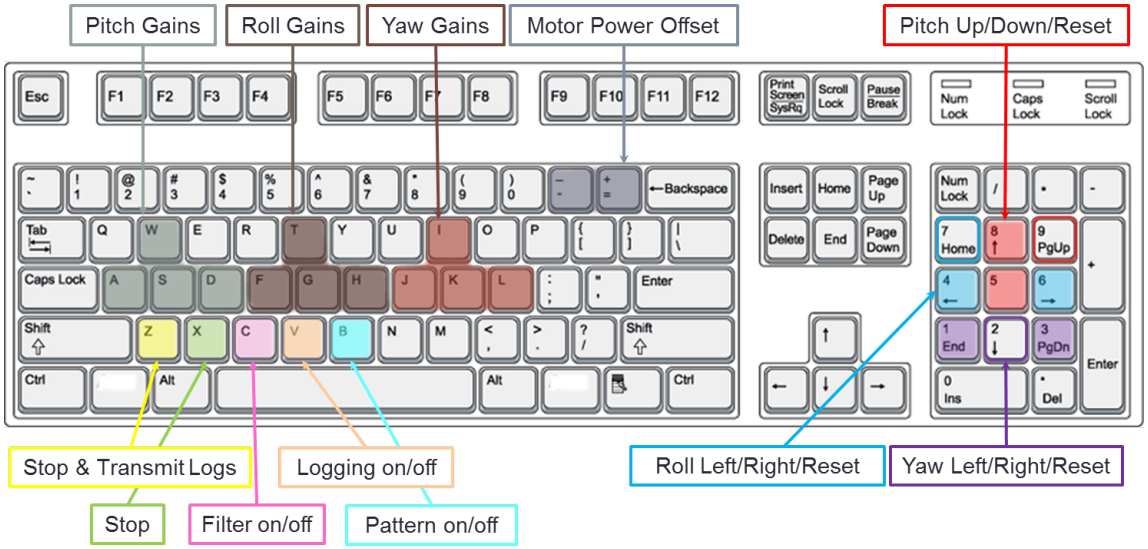



Figure 4.8: Keyboard commands

The reception section of the ground station (implemented in MATLAB, found in Appendix C) waits for the starting character ('B') and then reads packets from the reception transceiver via serial and stores them in a matrix. After receiving the terminating character ('G'), the ground station plots the data with respect to time in milliseconds. The data fields sent by the test bench are shown in Table 4.2.

## 4.5 Three Degree of Freedom Verification

### 4.5.1 Overview

We test the three degree of freedom capabilities of the test bench by analyzing its response to a sequence of commanded poses. Each pose in the sequence lasts three seconds and

Table 4.2: Packet Data Fields

| Packet Data Fields |
|---|
| gyro-x |
| gyro-x |
| gyro-x |
| accel-y |
| accel-y |
| accel-y |
| battery-volts |
| seconds |
| milliseconds |
| reference-gyro-x |
| reference-gyro-y |
| reference-gyro-z |
| reference-accel-x |
| reference-accel-y |
| power front |
| power left |
| power front left / back right |
| half-max-power |
| packet-number |

proceeds as in Table 4.3:

Table 4.3: Pattern Sequence

| Time (s) | Pattern Sequence |
|---|---|
| 3 | Pitch Forward (pitch reference - 150) |
| 6 | Level (pitch reference) |
| 9 | Pitch Backward (pitch reference + 150) |
| 12 | Level (pitch reference) |
| 15 | Roll Right (roll reference +150) |
| 18 | Level (roll reference) |
| 21 | Roll Left (roll reference -150) |
| 24 | Level (roll reference) |
| 27 | Yaw Left (roll reference - 300) |
| 30 | Level (yaw reference) |
| 33 | Yaw Right (roll reference + 300) |
| 36 | Level (yaw reference) |

The sensor and reference commands during this sequence are recorded in the flight logs allowing MATLAB to plot the motion of the test bench.

### 4.5.2 Results

The accelerometer measurements showing the response of the test bench to the commanded reference step inputs are shown in Figures 4.9 and 4.10. In Figure 4.11, the z-axis gyroscope measurement is used to show the yawing motion of the vehicle since the z-axis accelerometer measures no change in the direction of gravitational acceleration as the vehicle yaws.

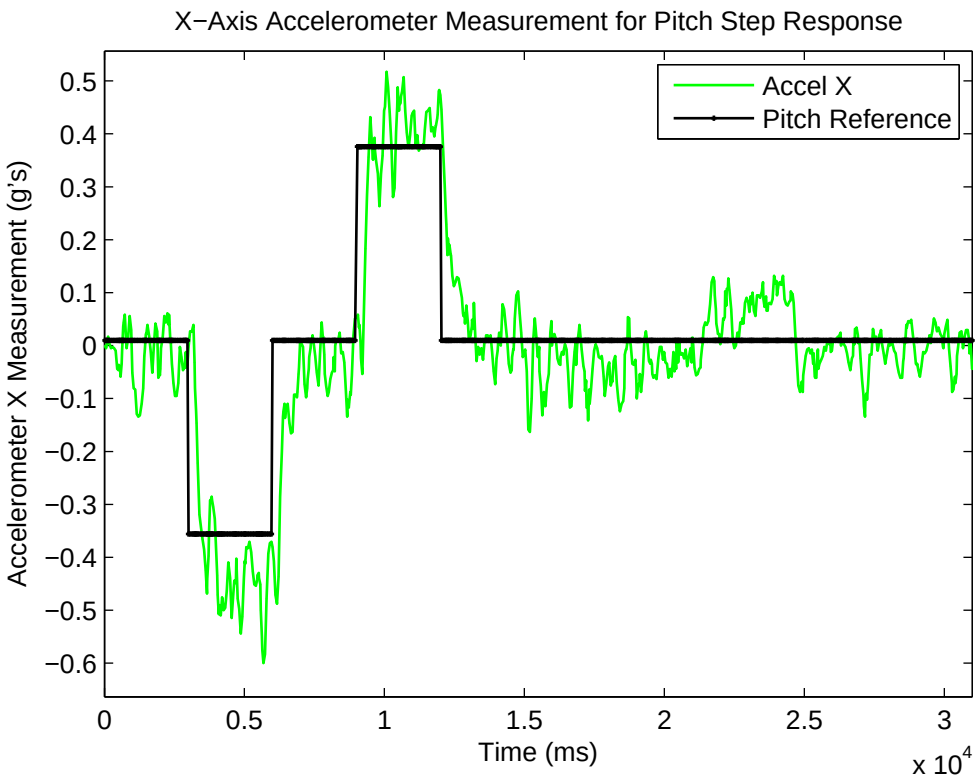



Figure 4.9: Step Responses in Pitch

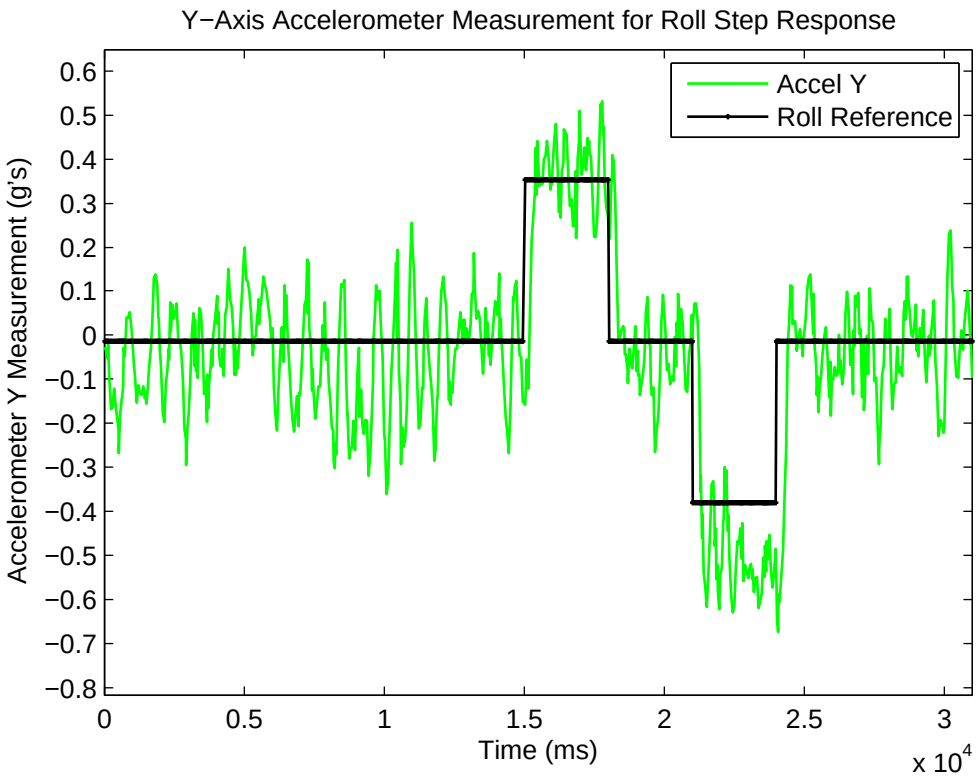



Figure 4.10: Step Responses in Roll

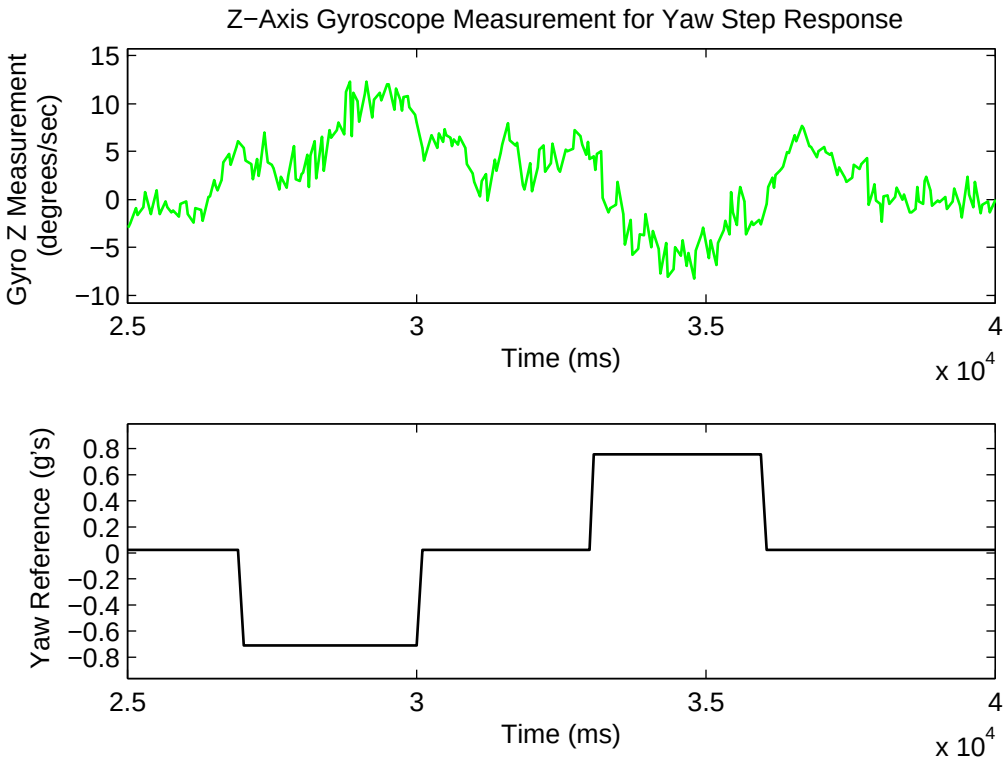



Figure 4.11: Step Response (Gyroscope) in Yaw

# CHAPTER V

# ANALYSIS AND CONCLUSION

In the simulation tests, the MAV model's roll, pitch, and yaw state variables converged to the step input and sinusoidal reference trajectories. The results of this test demonstrate that a nonlinear controller can control the three degrees of rotational freedom of our MAV model using the four wing maneuvering scheme defined in Section 1.3. In addition, we also verified that we can control the altitude in the model simultaneously with the pitch, roll, and yaw. We found a limitation in the ability of the model to track a sinusoidal reference trajectory with a frequency greater than ~0.4 Hz. This is because the vehicle's wings can only thrust downward (positive thrust) which reduces the vehicles agility. For example, if the vehicle's current altitude is greater than the reference altitude, the vehicle's downward acceleration is limited to its weight.

In the hardware tests on the test bench, both visual inspection and the accelerometer and gyroscope measurements showed the test bench rotating as desired to roll, pitch, and yaw commands. This is most apparent in the accelerometer measurements (Figures 4.9 and 4.10) which oscillated about the reference step values. The z-axis gyroscope measurements in Figure 4.11 show the rotational velocity about the z-axis increase for lower reference values and decrease for higher reference values. These tests confirm that a PD controller is capable of controlling the three degrees of rotational freedom of a physical vehicle using the four wing maneuvering scheme.

# APPENDIX A

# LINEARIZATION AND CONTROLLER MATRICES

This appendix includes the A, B, and U matrices shown in Equation (3.10) as displayed by MATLAB. It also includes the determinant of A from Equation (3.12). The best way to view the matrices is to execute the MATLAB code which finds these matrices and is given below.

```
% /*****************************************************************
% A Matrix calculated by Find_A.m
% *****************************************************************/
A =
[ (d*sin(x1)*tan(x2))/Iphi - (d*cos(x1)*tan(x2))/Ipsi - d/It, d/It +↙
(d*cos(x1)*tan(x2))/Ipsi + (d*sin(x1)*tan(x2))/Iphi, (d*cos(x1)*tan↙
(x2))/Ipsi - d/It - (d*sin(x1)*tan(x2))/Iphi, d/It - (d*cos(x1)*tan↙
(x2))/Ipsi - (d*sin(x1)*tan(x2))/Iphi]
[                        (d*cos(x1))/Iphi + (d*sin(x1))/Ipsi,↙
(d*cos(x1))/Iphi - (d*sin(x1))/Ipsi,                        - (d*cos↙
(x1))/Iphi - (d*sin(x1))/Ipsi,                       (d*sin(x1))/Ipsi↙
- (d*cos(x1))/Iphi]
[    (d*sin(x1))/(Iphi*cos(x2)) - (d*cos(x1))/(Ipsi*cos(x2)),↙
(d*cos(x1))/(Ipsi*cos(x2)) + (d*sin(x1))/(Iphi*cos(x2)),    (d*cos↙
(x1))/(Ipsi*cos(x2)) - (d*sin(x1))/(Iphi*cos(x2)),  - (d*cos(x1))/↙
(Ipsi*cos(x2)) - (d*sin(x1))/(Iphi*cos(x2))]
[                                      -(cos(x1)*cos(x2))/M,↙
-(cos(x1)*cos(x2))/M,                                      -(cos(x1)↙
*cos(x2))/M,                                    -(cos(x1)*cos(x2))↙
/M]

% /*****************************************************************
% Determinant of A calculated by Find_A.m
% *****************************************************************/
det(A) =
 (16*d^3*cos(x1))/(Iphi*Ipsi*It*M)

% /*****************************************************************
% B Matrix calculated by Find_B.m
% *****************************************************************/
B =
 (x5*cos(x1)*tan(x2) - x6*sin(x1)*tan(x2))*(x4 + x6*cos(x1)*tan(x2) +↙
x5*sin(x1)*tan(x2)) + ((x6*cos(x1))/cos(x2)^2 + (x5*sin(x1))/cos(x2)↙
^2)*(x5*cos(x1) - x6*sin(x1)) + (x5*x6*(Iphi - Ipsi))/It - (x4*x5*cos↙
(x1)*tan(x2)*(Iphi - It))/Ipsi + (x4*x6*sin(x1)*tan(x2)*(Ipsi - It))↙
/Iphi
↙
(x4*x6*cos(x1)*(Ipsi - It))/Iphi - (x6*cos(x1) + x5*sin(x1))*(x4 +↙
x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) + (x4*x5*sin(x1)*(Iphi - It))↙
/Ipsi
     (- x4*cos(x1)*Iphi^2*x5 + sin(2*x1)*tan(x2)*Iphi*Ipsi*x5^2 +↙
2*cos(2*x1)*tan(x2)*Iphi*Ipsi*x5*x6 + x4*cos(x1)*Iphi*Ipsi*x5 - sin↙
(2*x1)*tan(x2)*Iphi*Ipsi*x6^2 - x4*sin(x1)*Iphi*Ipsi*x6 + It*x4*cos↙
(x1)*Iphi*x5 + x4*sin(x1)*Ipsi^2*x6 - It*x4*sin(x1)*Ipsi*x6)/↙
(Iphi*Ipsi*cos(x2))
↙
G + cos(x1)*cos(x2)*(x10*x5 - x4*x9)
% /*****************************************************************
% U Matrix calculated by Find_U.m
```

```
% ************************************************************************/
 U =

 (It*((x5*cos(x1)*tan(x2) - x6*sin(x1)*tan(x2))*(x4 + x6*cos(x1)*tan↙
(x2) + x5*sin(x1)*tan(x2)) - y1d_dot_dot + kp1*(x1 - y1d) + ((x6*cos↙
(x1))/cos(x2)^2 + (x5*sin(x1))/cos(x2)^2)*(x5*cos(x1) - x6*sin(x1)) +↙
kd1*(x4 - y1d_dot + x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) + (x5*x6*↙
(Iphi - Ipsi))/It - (x4*x5*cos(x1)*tan(x2)*(Iphi - It))/Ipsi +↙
(x4*x6*sin(x1)*tan(x2)*(Ipsi - It))/Iphi))/(4*d) - ((It*sin(x2) -↙
Ipsi*cos(x1)*cos(x2) + Iphi*cos(x2)*sin(x1))*(kp3*(x3 - y3d) -↙
y3d_dot_dot + kd3*((x6*cos(x1))/cos(x2) - y3d_dot + (x5*sin(x1))/cos↙
(x2)) + (- x4*cos(x1)*Iphi^2*x5 + sin(2*x1)*tan(x2)*Iphi*Ipsi*x5^2 +↙
2*cos(2*x1)*tan(x2)*Iphi*Ipsi*x5*x6 + x4*cos(x1)*Iphi*Ipsi*x5 - sin↙
(2*x1)*tan(x2)*Iphi*Ipsi*x6^2 - x4*sin(x1)*Iphi*Ipsi*x6 + It*x4*cos↙
(x1)*Iphi*x5 + x4*sin(x1)*Ipsi^2*x6 - It*x4*sin(x1)*Ipsi*x6)/↙
(Iphi*Ipsi*cos(x2))))/(4*d) + ((Iphi*cos(x1) + Ipsi*sin(x1))*↙
(y2d_dot_dot + kd2*(y2d_dot - x5*cos(x1) + x6*sin(x1)) + (x6*cos(x1) +↙
x5*sin(x1))*(x4 + x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) - kp2*(x2 -↙
y2d) - (x4*x6*cos(x1)*(Ipsi - It))/Iphi - (x4*x5*sin(x1)*(Iphi - It))↙
/Ipsi))/(4*d) + (M*(G - y4d_dot_dot + kd4*(x8 - y4d_dot) + kp4*(x7 -↙
y4d) + cos(x1)*cos(x2)*(x10*x5 - x4*x9)))/(4*cos(x1)*cos(x2))
 ((Iphi*cos(x1) - Ipsi*sin(x1))*(y2d_dot_dot + kd2*(y2d_dot - x5*cos↙
(x1) + x6*sin(x1)) + (x6*cos(x1) + x5*sin(x1))*(x4 + x6*cos(x1)*tan↙
(x2) + x5*sin(x1)*tan(x2)) - kp2*(x2 - y2d) - (x4*x6*cos(x1)*(Ipsi -↙
It))/Iphi - (x4*x5*sin(x1)*(Iphi - It))/Ipsi))/(4*d) - ((Ipsi*cos(x1)↙
*cos(x2) - It*sin(x2) + Iphi*cos(x2)*sin(x1))*(kp3*(x3 - y3d) -↙
y3d_dot_dot + kd3*((x6*cos(x1))/cos(x2) - y3d_dot + (x5*sin(x1))/cos↙
(x2)) + (- x4*cos(x1)*Iphi^2*x5 + sin(2*x1)*tan(x2)*Iphi*Ipsi*x5^2 +↙
2*cos(2*x1)*tan(x2)*Iphi*Ipsi*x5*x6 + x4*cos(x1)*Iphi*Ipsi*x5 - sin↙
(2*x1)*tan(x2)*Iphi*Ipsi*x6^2 - x4*sin(x1)*Iphi*Ipsi*x6 + It*x4*cos↙
(x1)*Iphi*x5 + x4*sin(x1)*Ipsi^2*x6 - It*x4*sin(x1)*Ipsi*x6)/↙
(Iphi*Ipsi*cos(x2))))/(4*d) - (It*((x5*cos(x1)*tan(x2) - x6*sin(x1)↙
*tan(x2))*(x4 + x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) - y1d_dot_dot↙
+ kp1*(x1 - y1d) + ((x6*cos(x1))/cos(x2)^2 + (x5*sin(x1))/cos(x2)^2)*↙
(x5*cos(x1) - x6*sin(x1)) + kd1*(x4 - y1d_dot + x6*cos(x1)*tan(x2) +↙
x5*sin(x1)*tan(x2)) + (x5*x6*(Iphi - Ipsi))/It - (x4*x5*cos(x1)*tan↙
(x2)*(Iphi - It))/Ipsi + (x4*x6*sin(x1)*tan(x2)*(Ipsi - It))/Iphi))/↙
(4*d) + (M*(G - y4d_dot_dot + kd4*(x8 - y4d_dot) + kp4*(x7 - y4d) +↙
cos(x1)*cos(x2)*(x10*x5 - x4*x9)))/(4*cos(x1)*cos(x2))
 (It*((x5*cos(x1)*tan(x2) - x6*sin(x1)*tan(x2))*(x4 + x6*cos(x1)*tan↙
(x2) + x5*sin(x1)*tan(x2)) - y1d_dot_dot + kp1*(x1 - y1d) + ((x6*cos↙
(x1))/cos(x2)^2 + (x5*sin(x1))/cos(x2)^2)*(x5*cos(x1) - x6*sin(x1)) +↙
kd1*(x4 - y1d_dot + x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) + (x5*x6*↙
(Iphi - Ipsi))/It - (x4*x5*cos(x1)*tan(x2)*(Iphi - It))/Ipsi +↙
(x4*x6*sin(x1)*tan(x2)*(Ipsi - It))/Iphi))/(4*d) - ((It*sin(x2) +↙
Ipsi*cos(x1)*cos(x2) - Iphi*cos(x2)*sin(x1))*(kp3*(x3 - y3d) -↙
y3d_dot_dot + kd3*((x6*cos(x1))/cos(x2) - y3d_dot + (x5*sin(x1))/cos↙
(x2)) + (- x4*cos(x1)*Iphi^2*x5 + sin(2*x1)*tan(x2)*Iphi*Ipsi*x5^2 +↙
2*cos(2*x1)*tan(x2)*Iphi*Ipsi*x5*x6 + x4*cos(x1)*Iphi*Ipsi*x5 - sin↙
```

```
(2*x1)*tan(x2)*Iphi*Ipsi*x6^2 - x4*sin(x1)*Iphi*Ipsi*x6 + It*x4*cos↙
(x1)*Iphi*x5 + x4*sin(x1)*Ipsi^2*x6 - It*x4*sin(x1)*Ipsi*x6)/↙
(Iphi*Ipsi*cos(x2))))/(4*d) - ((Iphi*cos(x1) + Ipsi*sin(x1))*↙
(y2d_dot_dot + kd2*(y2d_dot - x5*cos(x1) + x6*sin(x1)) + (x6*cos(x1) +↙
x5*sin(x1))*(x4 + x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) - kp2*(x2 -↙
y2d) - (x4*x6*cos(x1)*(Ipsi - It))/Iphi - (x4*x5*sin(x1)*(Iphi - It))↙
/Ipsi))/(4*d) + (M*(G - y4d_dot_dot + kd4*(x8 - y4d_dot) + kp4*(x7 -↙
y4d) + cos(x1)*cos(x2)*(x10*x5 - x4*x9)))/(4*cos(x1)*cos(x2))
 ((It*sin(x2) + Ipsi*cos(x1)*cos(x2) + Iphi*cos(x2)*sin(x1))*(kp3*(x3↙
- y3d) - y3d_dot_dot + kd3*((x6*cos(x1))/cos(x2) - y3d_dot + (x5*sin↙
(x1))/cos(x2)) + (- x4*cos(x1)*Iphi^2*x5 + sin(2*x1)*tan(x2)↙
*Iphi*Ipsi*x5^2 + 2*cos(2*x1)*tan(x2)*Iphi*Ipsi*x5*x6 + x4*cos(x1)↙
*Iphi*Ipsi*x5 - sin(2*x1)*tan(x2)*Iphi*Ipsi*x6^2 - x4*sin(x1)↙
*Iphi*Ipsi*x6 + It*x4*cos(x1)*Iphi*x5 + x4*sin(x1)*Ipsi^2*x6 -↙
It*x4*sin(x1)*Ipsi*x6)/(Iphi*Ipsi*cos(x2))))/(4*d) - (It*((x5*cos(x1)↙
*tan(x2) - x6*sin(x1)*tan(x2))*(x4 + x6*cos(x1)*tan(x2) + x5*sin(x1)↙
*tan(x2)) - y1d_dot_dot + kp1*(x1 - y1d) + ((x6*cos(x1))/cos(x2)^2 +↙
(x5*sin(x1))/cos(x2)^2)*(x5*cos(x1) - x6*sin(x1)) + kd1*(x4 - y1d_dot↙
+ x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) + (x5*x6*(Iphi - Ipsi))/It↙
- (x4*x5*cos(x1)*tan(x2)*(Iphi - It))/Ipsi + (x4*x6*sin(x1)*tan(x2)*↙
(Ipsi - It))/Iphi))/(4*d) - ((Iphi*cos(x1) - Ipsi*sin(x1))*↙
(y2d_dot_dot + kd2*(y2d_dot - x5*cos(x1) + x6*sin(x1)) + (x6*cos(x1) +↙
x5*sin(x1))*(x4 + x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) - kp2*(x2 -↙
y2d) - (x4*x6*cos(x1)*(Ipsi - It))/Iphi - (x4*x5*sin(x1)*(Iphi - It))↙
/Ipsi))/(4*d) + (M*(G - y4d_dot_dot + kd4*(x8 - y4d_dot) + kp4*(x7 -↙
y4d) + cos(x1)*cos(x2)*(x10*x5 - x4*x9)))/(4*cos(x1)*cos(x2))
```

```
% /*****************************************************************
% * FileName:        find_A.m
% * Compiler:        MATLAB
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * Author            Date      Comments on this revision
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * David Smith       2011      Calculates A matrix for feedback lin.
% *****************************************************************/

clear all

syms('x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'r',↙
'I', 'M', 'G', 'd', 'Iphi', 'It', 'Ipsi')
%x9 and x10 are 'v' and 'u', the body frame linear velocities

f=[x4+x5*tan(x2)*sin(x1)+x6*tan(x2)*cos(x1);...
    x5*cos(x1)-x6*sin(x1);...
    x5*sin(x1)/cos(x2)+x6*cos(x1)/cos(x2);...
    -(Ipsi-Iphi)*x5*x6/It;...
    -(It-Ipsi)*x4*x6/Iphi;...
    -(Iphi-It)*x4*x5/Ipsi;...
```

```
    x8;...
    G+cos(x1)*cos(x2)*(-x4*x9+x5*x10);];

g=[ 0 0 0 0;...
    0 0 0 0;...
    0 0 0 0;...
    -d/It d/It -d/It d/It;...
    d/Iphi d/Iphi -d/Iphi -d/Iphi;...
    -d/Ipsi d/Ipsi d/Ipsi -d/Ipsi;...
    0 0 0 0;...
    -cos(x2)*cos(x1)/M -cos(x2)*cos(x1)/M -cos(x2)*cos(x1)/M -cos(x2)↙
*cos(x1)/M];

g1 = g(:,1);
g2 = g(:,2);
g3 = g(:,3);
g4 = g(:,4);

dLfH1 = [x5*tan(x2)*cos(x1)-x6*tan(x2)*sin(x1) x5*(sec(x2)^2)*sin(x1)↙
+x6*(sec(x2)^2)*cos(x1) 0 1 tan(x2)*sin(x1) tan(x2)*cos(x1) 0 0];
Lg1LfH1 = dLfH1*g1;
Lg2LfH1 = dLfH1*g2;
Lg3LfH1 = dLfH1*g3;
Lg4LfH1 = dLfH1*g4;

dLfH2 = [-x5*sin(x1)-x6*cos(x1) 0 0 0 cos(x1) -sin(x1) 0 0];
Lg1LfH2 = dLfH2*g1;
Lg2LfH2 = dLfH2*g2;
Lg3LfH2 = dLfH2*g3;
Lg4LfH2 = dLfH2*g4;

dLfH3 = [x5*cos(x1)/cos(x2)-x6*sin(x1)/cos(x2) x5*sin(x1)*tan(x2)*sec↙
(x2)+x6*cos(x1)*tan(x2)*sec(x2) 0 0 sin(x1)/cos(x2) cos(x1)/cos(x2) 0↙
0];
Lg1LfH3 = dLfH3*g1;
Lg2LfH3 = dLfH3*g2;
Lg3LfH3 = dLfH3*g3;
Lg4LfH3 = dLfH3*g4;

dLfH4 = [0 0 0 0 0 0 0 1];
Lg1LfH4 = dLfH4*g1;
Lg2LfH4 = dLfH4*g2;
Lg3LfH4 = dLfH4*g3;
Lg4LfH4 = dLfH4*g4;

A = [Lg1LfH1 Lg2LfH1 Lg3LfH1 Lg4LfH1;...
     Lg1LfH2 Lg2LfH2 Lg3LfH2 Lg4LfH2;...
     Lg1LfH3 Lg2LfH3 Lg3LfH3 Lg4LfH3;...
     Lg1LfH4 Lg2LfH4 Lg3LfH4 Lg4LfH4]
```

```
pretty(simplify(det(A)))

A_inv = inv(A)

% /*****************************************************************
% * FileName:        find_B.m
% * Compiler:        MATLAB
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * Author            Date      Comments on this revision
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * David Smith       2011      Calculates B matrix for feedback lin.
% *****************************************************************/

clear all

syms('x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'r',↙
'I', 'M', 'G', 'Iphi', 'It', 'Ipsi')

f=[x4+x5*tan(x2)*sin(x1)+x6*tan(x2)*cos(x1);...
    x5*cos(x1)-x6*sin(x1);...
    x5*sin(x1)/cos(x2)+x6*cos(x1)/cos(x2);...
    -(Ipsi-Iphi)*x5*x6/It;...
    -(It-Ipsi)*x4*x6/Iphi;...
    -(Iphi-It)*x4*x5/Ipsi;...
    x8;...
    G+cos(x1)*cos(x2)*(-x4*x9+x5*x10);];

dLfH1 = [x5*tan(x2)*cos(x1)-x6*tan(x2)*sin(x1) x5*(sec(x2)^2)*sin(x1)↙
+x6*(sec(x2)^2)*cos(x1) 0 1 tan(x2)*sin(x1) tan(x2)*cos(x1) 0 0];
Lf2H1 = simplify(dLfH1*f)

dLfH2 = [-x5*sin(x1)-x6*cos(x1) 0 0 0 cos(x1) -sin(x1) 0 0];
Lf2H2 = simplify(dLfH2*f)

dLfH3 = [x5*cos(x1)/cos(x2)-x6*sin(x1)/cos(x2) x5*sin(x1)*tan(x2)*sec↙
(x2)+x6*cos(x1)*tan(x2)*sec(x2) 0 0 sin(x1)/cos(x2) cos(x1)/cos(x2) 0↙
0];
Lf2H3 = simplify(dLfH3*f)

dLfH4 = [0 0 0 0 0 0 0 1];
Lf2H4 = simplify(dLfH4*f)

B = [Lf2H1; Lf2H2; Lf2H3; Lf2H4]

% /*****************************************************************
% * FileName:        find_U.m
% * Compiler:        MATLAB
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

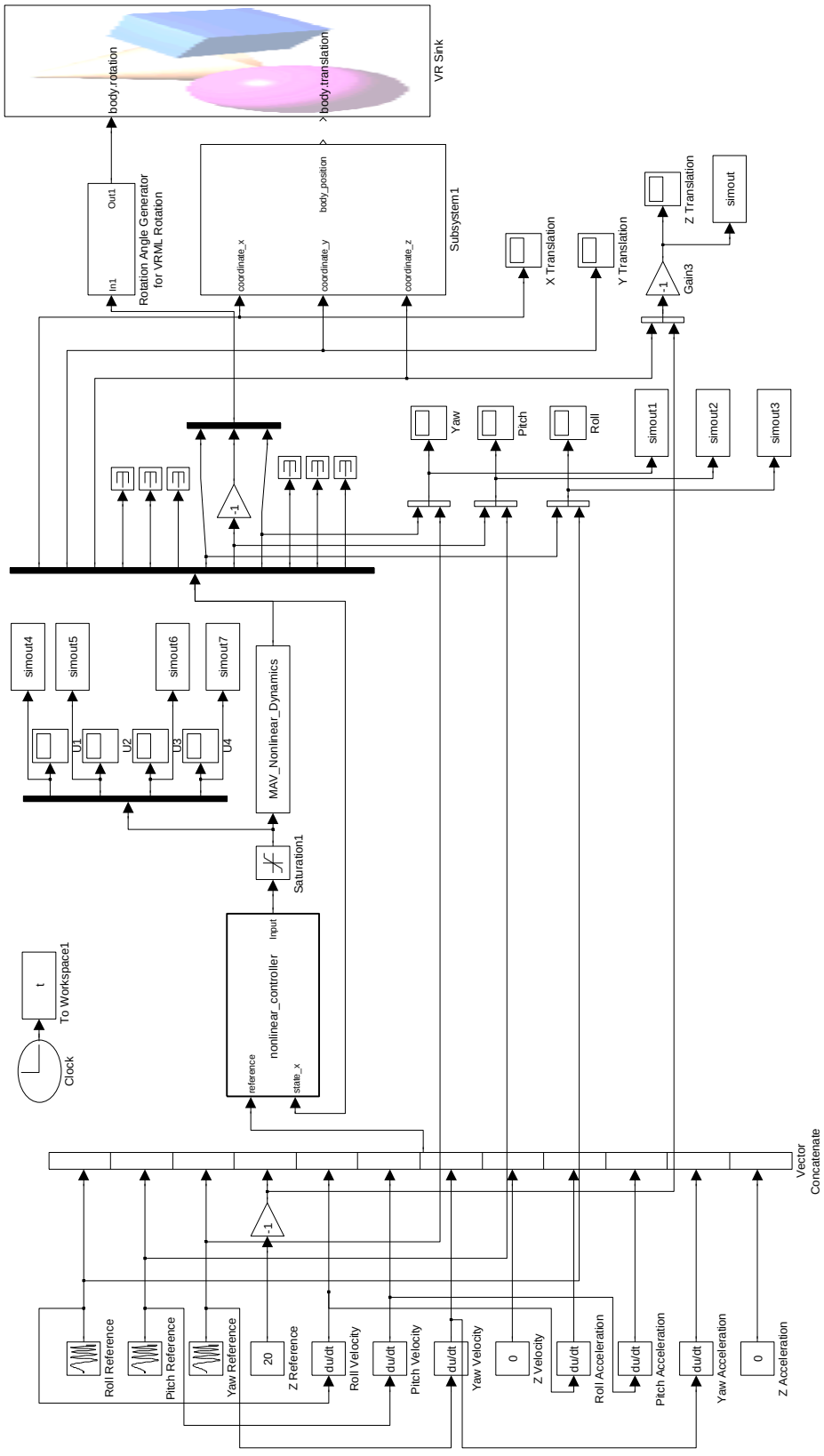
```
% * Author            Date      Comments on this revision
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * David Smith       2011      Calculates U matrix for feedback lin.
% ************************************************************************/

clear all

syms('x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'r',↙
'I', 'M', 'G', 'y1d_dot_dot', 'y1d_dot', 'y1d', 'y2d_dot_dot', 'Iphi',↙
'It', 'Ipsi', 'd',...
    'y2d_dot', 'y2d', 'y3d_dot_dot', 'y3d_dot', 'y3d', 'y4d_dot_dot',↙
'y4d_dot', 'y4d', 'kd1', 'kp1', 'kd2', 'kp2', 'kd3', 'kp3', 'kd4',↙
'kp4')

%as calculated by 'findA.m' and 'findB.m'
A_inv = [ -It/(4*d),  (Iphi*cos(x1) + Ipsi*sin(x1))/(4*d),  (It*sin↙
(x2) - Ipsi*cos(x1)*cos(x2) + Iphi*cos(x2)*sin(x1))/(4*d), -M/(4*cos↙
(x1)*cos(x2));
           It/(4*d),  (Iphi*cos(x1) - Ipsi*sin(x1))/(4*d),  (Ipsi*cos↙
(x1)*cos(x2) - It*sin(x2) + Iphi*cos(x2)*sin(x1))/(4*d), -M/(4*cos(x1)↙
*cos(x2));
          -It/(4*d), -(Iphi*cos(x1) + Ipsi*sin(x1))/(4*d),  (It*sin↙
(x2) + Ipsi*cos(x1)*cos(x2) - Iphi*cos(x2)*sin(x1))/(4*d), -M/(4*cos↙
(x1)*cos(x2));
           It/(4*d), -(Iphi*cos(x1) - Ipsi*sin(x1))/(4*d), -(It*sin↙
(x2) + Ipsi*cos(x1)*cos(x2) + Iphi*cos(x2)*sin(x1))/(4*d), -M/(4*cos↙
(x1)*cos(x2))]

B = [
 (x5*cos(x1)*tan(x2) - x6*sin(x1)*tan(x2))*(x4 + x6*cos(x1)*tan(x2) +↙
x5*sin(x1)*tan(x2)) + ((x6*cos(x1))/cos(x2)^2 + (x5*sin(x1))/cos(x2)↙
^2)*(x5*cos(x1) - x6*sin(x1)) + (x5*x6*(Iphi - Ipsi))/It - (x4*x5*cos↙
(x1)*tan(x2)*(Iphi - It))/Ipsi + (x4*x6*sin(x1)*tan(x2)*(Ipsi - It))↙
/Iphi;
↙
(x4*x6*cos(x1)*(Ipsi - It))/Iphi - (x6*cos(x1) + x5*sin(x1))*(x4 +↙
x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) + (x4*x5*sin(x1)*(Iphi - It))↙
/Ipsi;
     (- x4*cos(x1)*Iphi^2*x5 + sin(2*x1)*tan(x2)*Iphi*Ipsi*x5^2 +↙
2*cos(2*x1)*tan(x2)*Iphi*Ipsi*x5*x6 + x4*cos(x1)*Iphi*Ipsi*x5 - sin↙
(2*x1)*tan(x2)*Iphi*Ipsi*x6^2 - x4*sin(x1)*Iphi*Ipsi*x6 + It*x4*cos↙
(x1)*Iphi*x5 + x4*sin(x1)*Ipsi^2*x6 - It*x4*sin(x1)*Ipsi*x6)/↙
(Iphi*Ipsi*cos(x2));
↙
G + cos(x1)*cos(x2)*(x10*x5 - x4*x9)]

y1_dot_dot = y1d_dot_dot - kd1*(x4+x5*tan(x2)*sin(x1)+x6*tan(x2)*cos↙
(x1)-y1d_dot)-kp1*(x1-y1d);
y2_dot_dot = y2d_dot_dot - kd2*(x5*cos(x1)-x6*sin(x1)-y2d_dot)-kp2*↙
```

```
(x2-y2d);
y3_dot_dot = y3d_dot_dot - kd3*(x5*sin(x1)/cos(x2)+x6*cos(x1)/cos(x2)-↙
y3d_dot)-kp3*(x3-y3d);
y4_dot_dot = y4d_dot_dot - kd4*(x8-y4d_dot)-kp4*(x7-y4d);

Y = [y1_dot_dot; y2_dot_dot; y3_dot_dot; y4_dot_dot]

U = A_inv*(Y-B)

pretty(U)
```

# APPENDIX B

# MATLAB AND SIMULINK IMPLEMENTATION

This appendix includes the MATLAB simulation model and code to produce the results shown in Figure 3.3.


VR Sink
body rotation
body translation
Rotation Angle Generator for VRML Rotation
Out1
In1
Subsystem1
body_position
coordinate_x
coordinate_y
coordinate_z
X Translation
Y Translation
Z Translation
Gain3
simout
Yaw
Pitch
Roll
simout1
simout2
simout3
simout4
simout5
simout6
simout7
U1
U2
U3
U4
MAV_Nonlinear_Dynamics
Saturation1
nonlinear_controller
reference
input
state_x
Clock
To Workspace1
t
Roll Reference
Pitch Reference
Yaw Reference
20
Z Reference
Roll Velocity
Pitch Velocity
Yaw Velocity
0
Z Velocity
Roll Acceleration
Pitch Acceleration
Yaw Acceleration
0
Z Acceleration
du/dt
Vector Concatenate

```
% /*****************************************************************
% * FileName:        MAV_Initialize.m
% * Compiler:        MATLAB
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * Author            Date      Comments on this revision
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * David Smith       2011      Initializes Simulation Constants
% *****************************************************************/

clear all;

global J_phi J_theta J_psi d_wing G M_body Scale PI
Scale = 0.001;
G = 9.81;
PI = 3.14;

M_Total = 16*Scale; %16 grams
M_body = M_Total;
l_body = 2.54*6 * Scale; % 6 inches
w_body = 2.54*4 * Scale;
h_body = 2.54*0.75 * Scale;
d_wing = l_body / 2;

J_phi = (1/12)*M_body*(l_body^2);
J_theta = (1/12)*M_body*(w_body^2);%(1/3)*sum(M3.*L.^2)
J_psi = J_theta + J_phi; % Perpendicular axis theorem


% /*****************************************************************
% * FileName:        MAV_Nonlinear_Dynamics.m
% * Compiler:        MATLAB
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * Author            Date      Comments on this revision
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * David Smith       2011      S-function calculates next state vec.
% *****************************************************************/

function [sys, x0] = MAV_Nonlinear_Dynamics(t,state_x,U,flag)

if abs(flag) == 1
    sys = MAV_Nonlinear_Model(state_x, U);

elseif abs(flag) == 3
    sys = real(state_x);

elseif flag == 0
    sys = [12;0;12;4;0;1;];
    x0 = [0;0;0;0;0;0;0;0;0;0;0;0];
    str =[];
```

```
    ts = [0 0];

else
    sys = [];

end;


% /*****************************************************************
% * FileName:        MAV_Nonlinear_Model.m
% * Compiler:        MATLAB
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * Author            Date      Comments on this revision
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * David Smith       2011      Calculates derivative of state vec.
% *****************************************************************/

function state_xdot = MAV_Nonlinear_Model(x, U)

global J_phi J_theta J_psi d_wing G M_body

state_xdot = [...
    cos(x(8))*cos(x(9))*x(4)-(cos(x(7))*sin(x(9))-cos(x(9))*sin(x(8))↙
*sin(x(7)))*x(5)+(sin(x(9))*sin(x(7))+cos(x(9))*sin(x(8))*cos(x(7)))*x↙
(6);...
    cos(x(8))*sin(x(9))*x(4)+(cos(x(9))*cos(x(7))+sin(x(8))*sin(x(9))↙
*sin(x(7)))*x(5)-(cos(x(9))*sin(x(7))-cos(x(7))*sin(x(8))*sin(x(9)))*x↙
(6);...
    -sin(x(8))*x(4)+cos(x(8))*sin(x(7))*x(5)+cos(x(8))*cos(x(7))*x↙
(6);...
    -G*sin(x(8))-x(11)*x(6)+x(12)*x(5);...
    G*cos(x(8))*sin(x(7))-x(12)*x(4)+x(10)*x(6);...
    G*cos(x(8))*cos(x(7))-x(10)*x(5)+x(11)*x(4)-(U(1)+U(2)+U(3)+U(4))↙
/M_body;...
    x(10)+sin(x(7))*tan(x(8))*x(11)+cos(x(7))*tan(x(8))*x(12);...
    cos(x(7))*x(11)-sin(x(7))*x(12);...
    x(11)*sin(x(7))/cos(x(8))+x(12)*cos(x(7))/cos(x(8));...
    -x(11)*x(12)*(J_psi-J_phi)/J_theta+d_wing*(U(2)+U(4)-U(3)-U(1))↙
/J_theta;...
    -x(10)*x(12)*(J_theta-J_psi)/J_phi+d_wing*(U(1)+U(2)-U(3)-U(4))↙
/J_phi;...
    -x(10)*x(11)*(J_phi-J_theta)/J_psi+d_wing*(-U(1)-U(4)+U(2)+U(3))↙
/J_psi;...
    ];


% /*****************************************************************
% * FileName:        nonlinear_controller.m (embedded matlab code)
% * Compiler:        MATLAB
```

```
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * Author            Date      Comments on this revision
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * David Smith       2011      Calculates input from state vec & ref.
% ************************************************************************/

function Input = nonlinear_controller(reference, state_x)

%renaming variables
y1d = reference(1);
y2d = reference(2);
y3d = reference(3);
y4d = reference(4);
y1d_dot = reference(5);
y2d_dot = reference(6);
y3d_dot = reference(7);
y4d_dot = reference(8);
y1d_dot_dot = 0;
y2d_dot_dot = 0;
y3d_dot_dot = 0;
y4d_dot_dot = 0;

x1 = state_x(7); %theta
x2 = state_x(8); %phi
x3 = state_x(9); %psi
x4 = state_x(10); %p
x5 = state_x(11); %q
x6 = state_x(12); %r
x7 = state_x(3); %z
x8 = -sin(state_x(8))*state_x(4)+cos(state_x(8))*sin(state_x(7))↙
*state_x(5)+cos(state_x(8))*cos(state_x(7))*state_x(6); %z_dot
x9 = state_x(5); %v
x10= state_x(4); %u

%constants% NOTE: If plant contants change, these must be changed↙
manually.
Scale = 0.001;
G = 9.81;
M = 16*Scale;
It = 1.3763e-007;
Iphi = 3.0968e-007;
Ipsi = 4.4731e-007;
d = 0.0076;

kd1 = 300;
kp1 = 10000;
kd2 = 310;
kp2 = 10100;
kd3 = 290;
```

```
kp3 = 10000;
kd4 = 300;
kp4 = 10000;

%as calculated by 'findU.m'
Input = [
 (It*((x5*cos(x1)*tan(x2) - x6*sin(x1)*tan(x2))*(x4 + x6*cos(x1)*tan↙
(x2) + x5*sin(x1)*tan(x2)) - y1d_dot_dot + kp1*(x1 - y1d) + ((x6*cos↙
(x1))/cos(x2)^2 + (x5*sin(x1))/cos(x2)^2)*(x5*cos(x1) - x6*sin(x1)) +↙
kd1*(x4 - y1d_dot + x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) + (x5*x6*↙
(Iphi - Ipsi))/It - (x4*x5*cos(x1)*tan(x2)*(Iphi - It))/Ipsi +↙
(x4*x6*sin(x1)*tan(x2)*(Ipsi - It))/Iphi))/(4*d) - ((It*sin(x2) -↙
Ipsi*cos(x1)*cos(x2) + Iphi*cos(x2)*sin(x1))*(kp3*(x3 - y3d) -↙
y3d_dot_dot + kd3*((x6*cos(x1))/cos(x2) - y3d_dot + (x5*sin(x1))/cos↙
(x2)) + (- x4*cos(x1)*Iphi^2*x5 + sin(2*x1)*tan(x2)*Iphi*Ipsi*x5^2 +↙
2*cos(2*x1)*tan(x2)*Iphi*Ipsi*x5*x6 + x4*cos(x1)*Iphi*Ipsi*x5 - sin↙
(2*x1)*tan(x2)*Iphi*Ipsi*x6^2 - x4*sin(x1)*Iphi*Ipsi*x6 + It*x4*cos↙
(x1)*Iphi*x5 + x4*sin(x1)*Ipsi^2*x6 - It*x4*sin(x1)*Ipsi*x6)/↙
(Iphi*Ipsi*cos(x2))))/(4*d) + ((Iphi*cos(x1) + Ipsi*sin(x1))*↙
(y2d_dot_dot + kd2*(y2d_dot - x5*cos(x1) + x6*sin(x1)) + (x6*cos(x1) +↙
x5*sin(x1))*(x4 + x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) - kp2*(x2 -↙
y2d) - (x4*x6*cos(x1)*(Ipsi - It))/Iphi - (x4*x5*sin(x1)*(Iphi - It))↙
/Ipsi))/(4*d) + (M*(G - y4d_dot_dot + kd4*(x8 - y4d_dot) + kp4*(x7 -↙
y4d) + cos(x1)*cos(x2)*(x10*x5 - x4*x9)))/(4*cos(x1)*cos(x2));
 ((Iphi*cos(x1) - Ipsi*sin(x1))*(y2d_dot_dot + kd2*(y2d_dot - x5*cos↙
(x1) + x6*sin(x1)) + (x6*cos(x1) + x5*sin(x1))*(x4 + x6*cos(x1)*tan↙
(x2) + x5*sin(x1)*tan(x2)) - kp2*(x2 - y2d) - (x4*x6*cos(x1)*(Ipsi -↙
It))/Iphi - (x4*x5*sin(x1)*(Iphi - It))/Ipsi))/(4*d) - ((Ipsi*cos(x1)↙
*cos(x2) - It*sin(x2) + Iphi*cos(x2)*sin(x1))*(kp3*(x3 - y3d) -↙
y3d_dot_dot + kd3*((x6*cos(x1))/cos(x2) - y3d_dot + (x5*sin(x1))/cos↙
(x2)) + (- x4*cos(x1)*Iphi^2*x5 + sin(2*x1)*tan(x2)*Iphi*Ipsi*x5^2 +↙
2*cos(2*x1)*tan(x2)*Iphi*Ipsi*x5*x6 + x4*cos(x1)*Iphi*Ipsi*x5 - sin↙
(2*x1)*tan(x2)*Iphi*Ipsi*x6^2 - x4*sin(x1)*Iphi*Ipsi*x6 + It*x4*cos↙
(x1)*Iphi*x5 + x4*sin(x1)*Ipsi^2*x6 - It*x4*sin(x1)*Ipsi*x6)/↙
(Iphi*Ipsi*cos(x2))))/(4*d) - (It*((x5*cos(x1)*tan(x2) - x6*sin(x1)↙
*tan(x2))*(x4 + x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) - y1d_dot_dot↙
+ kp1*(x1 - y1d) + ((x6*cos(x1))/cos(x2)^2 + (x5*sin(x1))/cos(x2)^2)*↙
(x5*cos(x1) - x6*sin(x1)) + kd1*(x4 - y1d_dot + x6*cos(x1)*tan(x2) +↙
x5*sin(x1)*tan(x2)) + (x5*x6*(Iphi - Ipsi))/It - (x4*x5*cos(x1)*tan↙
(x2)*(Iphi - It))/Ipsi + (x4*x6*sin(x1)*tan(x2)*(Ipsi - It))/Iphi))/↙
(4*d) + (M*(G - y4d_dot_dot + kd4*(x8 - y4d_dot) + kp4*(x7 - y4d) +↙
cos(x1)*cos(x2)*(x10*x5 - x4*x9)))/(4*cos(x1)*cos(x2));
 (It*((x5*cos(x1)*tan(x2) - x6*sin(x1)*tan(x2))*(x4 + x6*cos(x1)*tan↙
(x2) + x5*sin(x1)*tan(x2)) - y1d_dot_dot + kp1*(x1 - y1d) + ((x6*cos↙
(x1))/cos(x2)^2 + (x5*sin(x1))/cos(x2)^2)*(x5*cos(x1) - x6*sin(x1)) +↙
kd1*(x4 - y1d_dot + x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) + (x5*x6*↙
(Iphi - Ipsi))/It - (x4*x5*cos(x1)*tan(x2)*(Iphi - It))/Ipsi +↙
(x4*x6*sin(x1)*tan(x2)*(Ipsi - It))/Iphi))/(4*d) - ((It*sin(x2) +↙
Ipsi*cos(x1)*cos(x2) - Iphi*cos(x2)*sin(x1))*(kp3*(x3 - y3d) -↙
```

```
y3d_dot_dot + kd3*((x6*cos(x1))/cos(x2) - y3d_dot + (x5*sin(x1))/cos↙
(x2)) + (- x4*cos(x1)*Iphi^2*x5 + sin(2*x1)*tan(x2)*Iphi*Ipsi*x5^2 +↙
2*cos(2*x1)*tan(x2)*Iphi*Ipsi*x5*x6 + x4*cos(x1)*Iphi*Ipsi*x5 - sin↙
(2*x1)*tan(x2)*Iphi*Ipsi*x6^2 - x4*sin(x1)*Iphi*Ipsi*x6 + It*x4*cos↙
(x1)*Iphi*x5 + x4*sin(x1)*Ipsi^2*x6 - It*x4*sin(x1)*Ipsi*x6)/↙
(Iphi*Ipsi*cos(x2))))/(4*d) - ((Iphi*cos(x1) + Ipsi*sin(x1))*↙
(y2d_dot_dot + kd2*(y2d_dot - x5*cos(x1) + x6*sin(x1)) + (x6*cos(x1) +↙
x5*sin(x1))*(x4 + x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) - kp2*(x2 -↙
y2d) - (x4*x6*cos(x1)*(Ipsi - It))/Iphi - (x4*x5*sin(x1)*(Iphi - It))↙
/Ipsi))/(4*d) + (M*(G - y4d_dot_dot + kd4*(x8 - y4d_dot) + kp4*(x7 -↙
y4d) + cos(x1)*cos(x2)*(x10*x5 - x4*x9)))/(4*cos(x1)*cos(x2));
 ((It*sin(x2) + Ipsi*cos(x1)*cos(x2) + Iphi*cos(x2)*sin(x1))*(kp3*(x3↙
- y3d) - y3d_dot_dot + kd3*((x6*cos(x1))/cos(x2) - y3d_dot + (x5*sin↙
(x1))/cos(x2)) + (- x4*cos(x1)*Iphi^2*x5 + sin(2*x1)*tan(x2)↙
*Iphi*Ipsi*x5^2 + 2*cos(2*x1)*tan(x2)*Iphi*Ipsi*x5*x6 + x4*cos(x1)↙
*Iphi*Ipsi*x5 - sin(2*x1)*tan(x2)*Iphi*Ipsi*x6^2 - x4*sin(x1)↙
*Iphi*Ipsi*x6 + It*x4*cos(x1)*Iphi*x5 + x4*sin(x1)*Ipsi^2*x6 -↙
It*x4*sin(x1)*Ipsi*x6)/(Iphi*Ipsi*cos(x2))))/(4*d) - (It*((x5*cos(x1)↙
*tan(x2) - x6*sin(x1)*tan(x2))*(x4 + x6*cos(x1)*tan(x2) + x5*sin(x1)↙
*tan(x2)) - y1d_dot_dot + kp1*(x1 - y1d) + ((x6*cos(x1))/cos(x2)^2 +↙
(x5*sin(x1))/cos(x2)^2)*(x5*cos(x1) - x6*sin(x1)) + kd1*(x4 - y1d_dot↙
+ x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) + (x5*x6*(Iphi - Ipsi))/It↙
- (x4*x5*cos(x1)*tan(x2)*(Iphi - It))/Ipsi + (x4*x6*sin(x1)*tan(x2)*↙
(Ipsi - It))/Iphi))/(4*d) - ((Iphi*cos(x1) - Ipsi*sin(x1))*↙
(y2d_dot_dot + kd2*(y2d_dot - x5*cos(x1) + x6*sin(x1)) + (x6*cos(x1) +↙
x5*sin(x1))*(x4 + x6*cos(x1)*tan(x2) + x5*sin(x1)*tan(x2)) - kp2*(x2 -↙
y2d) - (x4*x6*cos(x1)*(Ipsi - It))/Iphi - (x4*x5*sin(x1)*(Iphi - It))↙
/Ipsi))/(4*d) + (M*(G - y4d_dot_dot + kd4*(x8 - y4d_dot) + kp4*(x7 -↙
y4d) + cos(x1)*cos(x2)*(x10*x5 - x4*x9)))/(4*cos(x1)*cos(x2))];


% /*****************************************************************
% * FileName:        Plotting.m
% * Compiler:        MATLAB
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * Author            Date      Comments on this revision
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * David Smith       2011      Plots vehicle roll, pitch, yaw, z
% *****************************************************************/

figure;
subplot(411),plot(t, simout3.signals.values(:,1),'g'),hold on;
subplot(411),plot(t, simout3.signals.values(:,2),'r--'),legend↙
('actual','reference');
subplot(412),plot(t, simout2.signals.values(:,1),'g'),hold on;
subplot(412),plot(t, simout2.signals.values(:,2),'r--');
subplot(413),plot(t, simout1.signals.values(:,1),'g'),hold on;
subplot(413),plot(t, simout1.signals.values(:,2),'r--');
subplot(414),plot(t, simout.signals.values(:,1),'g'),hold on;
```

```
subplot(414),plot(t, simout.signals.values(:,2),'r--');
subplot(411), ylabel('Roll (rad)'), xlabel('time [s]')
subplot(412), ylabel('Pitch (rad)'), xlabel('time [s]')
subplot(413), ylabel('Yaw (rad)'), xlabel('time [s]')
subplot(414), ylabel('Z (m)'), xlabel('time [s]')
hold off;
```

# APPENDIX C

# TEST BENCH IMPLEMENTATION

This appendix includes the ground station code used to receive flight logs from the test bench. It also gives all the control code from the test bench including main code, subroutines, and header files.

```
% /*****************************************************************
% * FileName:        MAV_GS_03.m
% * Compiler:        MATLAB
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * Author            Date      Comments on this revision
% *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% * David Smith       2011      Receives Flight Logs from MAV platform
% *****************************************************************/

clear;
data_per_row = 3; % number of sensor values per packet
each_data_len = 4; % Integer -> character length
separators_len = data_per_row; % space space semi-colon
packets_per_cycle = 3; %acc_x, acc_y, acc_z; gyro_x, gyro_y ... etc
freq_op_mav = 30; %Hz
operation_time = 60; %Hz

total_characters = (data_per_row * each_data_len + separators_len) *↙
packets_per_cycle * freq_op_mav * operation_time;

sport = serial('COM3', 'BaudRate', 115200);
set(sport, 'Timeout', 1000);
set(sport, 'InputBufferSize', 1000000);
fopen(sport);

value = fread(sport, 1);
while value ~= 'B'
    value = fread(sport, 1);
end

disp('B read... pausing');
pause(5);%sec

disp('getting data');

value = fread(sport, 1); %get past 'B'
 while value == 0
     value = fread(sport, 1); %get next
 end

str = zeros(1,4);
matrix = zeros(1000, 19);
k = 1;

while value ~= 'G' %keep repeating till the end of the packet is↙
received
j = 1;
 while value ~= ';'
     i = 1;
```

```
    while ~((value == ' ') || (value == ';'))
        str(i) = char(value);
        i = i + 1;
        value = fread(sport, 1);
    end

    if (value == ' ')
         value = fread(sport, 1);
    end

    c = sprintf('%c%c%c%c', str(1), str(2), str(3), str(4));
    matrix(k, j) = str2double(c);
    j = j + 1;
end

value = fread(sport, 1); %get past '\n'
value = fread(sport, 1); %get past '\r'
value = fread(sport, 1); %get next
while value == 0
    value = fread(sport, 1); %get next
end
matrix(k,:)
k = k + 1;
end

disp('read G');
if value == 'G'
    fclose(sport);
else
    disp('time out: the mav doesnt not respond');
end

start_frame = 1;
stop_frame = 730;
col_s = 8;
col_ms = 9;
title_list = {'gyro-x', 'gyro-y', 'gyro-z', 'accel-x', 'accel-y',↙
'accel-z', 'batt-volts', 'seconds', 'milliseconds', 'ref-gyro-x',↙
'ref-gyro-y', 'ref-gyro-z', 'ref-accel-x', 'ref-accel-y', 'power-↙
front', 'power-left', 'power-FL-BR', 'half-max-power', 'packet no'};
for j=1:19
    %if (j==2 || j==3 || j==5 || j==6 || j==11 || j==12 || j==14 ||↙
j==16 || 17)
    %    hold on;
    %if (j==1 || j==4 || j==5 || j==6 || j==7 || j==8 || j==9 || j==10
        figure;
    %    grid;
    %end
    time = matrix((start_frame:stop_frame),col_s)*1000 + matrix↙
```

```
((start_frame:stop_frame),col_ms);
    plot(time, matrix((start_frame:stop_frame),j));
    title(title_list{j});
end

% Gyro data
figure
j=1;%x
time = matrix((start_frame:stop_frame),col_s)*1000 + matrix↙
((start_frame:stop_frame),col_ms);
plot(time, matrix((start_frame:stop_frame),j),'g.-','LineWidth',↙
1,'MarkerFaceColor','g','MarkerSize',2);
hold on;
j=2;%y
plot(time, matrix((start_frame:stop_frame),j),'bo-','LineWidth',↙
1,'MarkerFaceColor','b','MarkerSize',2);
hold on;
j=3;%z
plot(time, matrix((start_frame:stop_frame),j),'r+-','LineWidth',↙
1,'MarkerFaceColor','r','MarkerSize',2);
title('Three Axis Gyros');
legend('x-Gyro','y-Gyro','z-Gyro');

% Gyro X data
figure
j=1;%x
time = matrix((start_frame:stop_frame),col_s)*1000 + matrix↙
((start_frame:stop_frame),col_ms);
plot(time, matrix((start_frame:stop_frame),j),'g.-','LineWidth',↙
1,'MarkerFaceColor','g','MarkerSize',2);
hold on;
j=10;%x-ref
plot(time, matrix((start_frame:stop_frame),j),'k+-','LineWidth',↙
1,'MarkerFaceColor','b','MarkerSize',2);
title('Gyro X');
legend('x-Gyro','x-Ref-Gyro');

% Gyro X data
figure
j=2;%y
time = matrix((start_frame:stop_frame),col_s)*1000 + matrix↙
((start_frame:stop_frame),col_ms);
plot(time, matrix((start_frame:stop_frame),j),'g.-','LineWidth',↙
1,'MarkerFaceColor','g','MarkerSize',2);
hold on;
j=11;%y-ref
plot(time, matrix((start_frame:stop_frame),j),'k+-','LineWidth',↙
1,'MarkerFaceColor','b','MarkerSize',2);
title('Gyro Y');
```

```
legend('y-Gyro','y-Ref-Gyro');

% Gyro Z data
figure
j=3;%z
time = matrix((start_frame:stop_frame),col_s)*1000 + matrix↙
((start_frame:stop_frame),col_ms);
plot(time, matrix((start_frame:stop_frame),j),'g.-','LineWidth',↙
1,'MarkerFaceColor','g','MarkerSize',2);
hold on;
j=12;%z-ref
plot(time, matrix((start_frame:stop_frame),j),'k+-','LineWidth',↙
1,'MarkerFaceColor','b','MarkerSize',2);
title('Gyro Z');
legend('Z-Gyro','z-Ref-Gyro');

% Accel data
figure
j=4;%x
time = matrix((start_frame:stop_frame),col_s)*1000 + matrix↙
((start_frame:stop_frame),col_ms);
plot(time, matrix((start_frame:stop_frame),j),'g.-','LineWidth',↙
1,'MarkerFaceColor','g','MarkerSize',2);
hold on;
j=5;%y
plot(time, matrix((start_frame:stop_frame),j),'bo-','LineWidth',↙
1,'MarkerFaceColor','b','MarkerSize',2);
hold on;
j=6;%z
plot(time, matrix((start_frame:stop_frame),j),'r+-','LineWidth',↙
1,'MarkerFaceColor','r','MarkerSize',2);
title('Three Axis Accels');
legend('x-Accel','y-Accel','z-Accel');

% Accel X data
figure
j=4;%x
time = matrix((start_frame:stop_frame),col_s)*1000 + matrix↙
((start_frame:stop_frame),col_ms);
plot(time, matrix((start_frame:stop_frame),j),'g.-','LineWidth',↙
1,'MarkerFaceColor','g','MarkerSize',2);
hold on;
j=13;%x-ref
plot(time, matrix((start_frame:stop_frame),j),'k+-','LineWidth',↙
1,'MarkerFaceColor','b','MarkerSize',2);
title('Accel X');
legend('X-Accel','x-Ref-Accel');

% Accel Y data
```

```
figure
j=5;%y
time = matrix((start_frame:stop_frame),col_s)*1000 + matrix↙
((start_frame:stop_frame),col_ms);
plot(time, matrix((start_frame:stop_frame),j),'g.-','LineWidth',↙
1,'MarkerFaceColor','g','MarkerSize',2);
hold on;
j=14;%y-ref
plot(time, matrix((start_frame:stop_frame),j),'k+-','LineWidth',↙
1,'MarkerFaceColor','b','MarkerSize',2);
title('Accel Y');
legend('Y-Accel','y-Ref-Accel');

% Power data
figure
j=15;%front
time = matrix((start_frame:stop_frame),col_s)*1000 + matrix↙
((start_frame:stop_frame),col_ms);
plot(time, matrix((start_frame:stop_frame),j),'g.-','LineWidth',↙
1,'MarkerFaceColor','g','MarkerSize',2);
hold on;
j=16;%left
plot(time, matrix((start_frame:stop_frame),j),'bo-','LineWidth',↙
1,'MarkerFaceColor','b','MarkerSize',2);
hold on;
j=17;%right
plot(time, matrix((start_frame:stop_frame),j),'r+-','LineWidth',↙
1,'MarkerFaceColor','r','MarkerSize',2);
hold on;
j=17;%half
plot(time, matrix((start_frame:stop_frame),j),'kd-','LineWidth',↙
1,'MarkerFaceColor','k','MarkerSize',2);
title('Output Powers');
legend('P-front','P-left','P-FL-BR','P-half');
```

```
/*********************************************************************
* FileName:        main_rtc.c
* Dependencies:    p33FJ256GP710.h, math.h, delay.h, xceiver_init.h
* Processor:       dsPIC33F
* Compiler:        MPLAB® C30 v2.01 or higher
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Author             Date      Comments on this revision
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* David Smith        2011     Main Code for MAV
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                                        Version: 1.23
*********************************************************************/

#include "p33FJ256GP710A.h"
#include "math.h"
#include "delay.h"
#include "xceiver_init.h"

_FOSCSEL(FNOSC_PRIPLL); //use primary oscillator with PLL
_FOSC(FCKSM_CSDCMD & OSCIOFNC_OFF & POSCMD_XT);
_FWDT(FWDTEN_OFF);


////////////////DEFINE GLOBAL VARIABLES//////////////////

#define PI 3.14159
#define BLOCK_LENGTH 12
#define g_X_ref 1725
#define g_Y_ref 1725
#define g_Z_ref 2150
#define a_X_ref 2050
#define a_Y_ref 2015
#define a_Z_ref 2404

const long taps[BLOCK_LENGTH]={767, 581, 740, 880, 983, 1037, 1037, 983, 880, 740,
581, 767};

unsigned int DmaBuffer = 0;
int ADC_Done = 0, new_cycle = 0, end = 0, cycle_num = 0, pattern_on_off = 0,
seconds = 0, milliseconds = 0,
       filter_on_off = 0, logging_on_off = 0, pattern_num = 0, command_mult = 3,
       patt_hold_time = 3, master_pattern_on_off = 1;
int full_max_power = 500; //scale is 0-500
int half_max_power = 350;
const int max_cycle_num = 736; //max number of sets of 19 bytes that can fit into
Data_Array buffer
const char packet_size = 19;

double step_size = 0.05;

double pk = 0.1;  //0.125 previously
double rk = 0.1;
double yk = 0.5;

int powerfront = 100;
int powerback = 100;
int powerleft = 100;
int powerright = 100;
```

```
int powerFL = 100;
int powerFR = 100;
int powerBL = 100;
int powerBR = 100;
int    powerFL_BR = 100;
int    powerFR_BL = 100;

typedef struct
{
	long inputstream[BLOCK_LENGTH];
	long output;
	int index;
} FilterData;

FilterData accel_X_data = {{a_X_ref, a_X_ref, a_X_ref, a_X_ref, a_X_ref, a_X_ref,
a_X_ref,
a_X_ref, a_X_ref, a_X_ref, a_X_ref}, 0, 0};

FilterData accel_Y_data = {{a_Y_ref, a_Y_ref, a_Y_ref, a_Y_ref, a_Y_ref, a_Y_ref,
a_Y_ref,
a_Y_ref, a_Y_ref, a_Y_ref, a_Y_ref}, 0, 0};

FilterData accel_Z_data = {{a_Z_ref, a_Z_ref, a_Z_ref, a_Z_ref, a_Z_ref, a_Z_ref,
a_Z_ref,
a_Z_ref, a_Z_ref, a_Z_ref, a_Z_ref}, 0, 0};

FilterData gyro_X_data = {{g_X_ref, g_X_ref, g_X_ref, g_X_ref, g_X_ref, g_X_ref,
g_X_ref,
g_X_ref, g_X_ref, g_X_ref, g_X_ref}, 0, 0};

FilterData gyro_Y_data = {{g_Y_ref, g_Y_ref, g_Y_ref, g_Y_ref, g_Y_ref, g_Y_ref,
g_Y_ref,
g_Y_ref, g_Y_ref, g_Y_ref, g_Y_ref}, 0, 0};

FilterData gyro_Z_data = {{g_Z_ref, g_Z_ref, g_Z_ref, g_Z_ref, g_Z_ref, g_Z_ref,
g_Z_ref,
g_Z_ref, g_Z_ref, g_Z_ref, g_Z_ref}, 0, 0};

typedef struct
{
	int gyro_X;
	int gyro_Y;
	int gyro_Z;
	int accel_X;
	int accel_Y;
	int accel_Z;
	int Batt_voltage;
	int Voltage33;
	int VDD_Voltage;
} SensorData;

float omega[3] = {0};
float accel[3] = {0};
float euler[3] = {0};

SensorData average = {0}, current = {0}, reference = {0}, error = {0};

typedef struct
```

```
{
	double Kp;
	double Kd;
	double Ki;
} Gains;

Gains pitch = {2.05, 0.5, 0};
Gains roll =  {0.85, 0.1, 0};
Gains yaw =   {1, .35, 0};//PID gains

unsigned int BufferA[9] __attribute__((space(dma))); // 9 ADC inputs

//////////////////FUNCTION DECLARATIONS//////////////////

void __builtin_btg(unsigned int *, unsigned int);
void calibrate(void);
void commands(char dir);
void Init_ADC_multi_input( void );
void initDma1(void);
void init_PWM(void);
void Init_Timer45( void );
void setPWM(int FrontLeft, int FrontRight, int RearLeft, int RearRight);
void init_SPI( void );
void batt_voltage_check();
void transmit_log(const unsigned int * buff, char * payload_buffer);
void saturate_at_limits(int * value, int low, int hi);
void saturate_at_limits_double(double * value, double low, double hi);
void log_data(unsigned int * buff);
int convolution(FilterData * object, const int new_element);
void next_pattern( void );

int main ( void )
{
	///////////////////INITIALIZE VARIABLES//////////////////

	unsigned int Data_Array[14000];

	//initial motor power

	int pitch_cont_input;
	int roll_cont_input;
	int yaw_cont_input;

	char payload_buffer[32 + 1];
	flush_buffer(payload_buffer);

	char direction = 0;

	//////////////////SET UP HARDWARE//////////////////

	DMA1STA = __builtin_dmaoffset(BufferA);

	TRISGbits.TRISG1 = 0;	// LED D1 for cycle start
	PORTGbits.RG1 = 1;
	TRISGbits.TRISG13 = 0;	// LED D4 for data received
	PORTGbits.RG13 = 1;
	TRISFbits.TRISF0 = 0;	// red LED D5 for low voltage warning (on
at/below ~3.0V)
```

```
PORTFbits.RF0 = 1;
TRISGbits.TRISG0 = 0;       // LED D2 for data log full/transmission
PORTGbits.RG0 = 1;
TRISGbits.TRISG12 = 0;      // LED D3 not really used
PORTGbits.RG12 = 1;

init_SPI();
init_transceiver();
initDma1();
Init_ADC_multi_input();
Init_Timer1();
Init_Timer4();

__builtin_btg((unsigned int *)&LATG, 12);
Delay(Delay_1S_Cnt);
__builtin_btg((unsigned int *)&LATG, 12);
Delay(Delay_1S_Cnt);
__builtin_btg((unsigned int *)&LATG, 12);
Delay(Delay_1S_Cnt);
__builtin_btg((unsigned int *)&LATG, 12);
Delay(Delay_1S_Cnt);

PORTGbits.RG12 = 0;
calibrate();
PORTGbits.RG12 = 1;

init_PWM();

reference.gyro_X = average.gyro_X;
reference.gyro_Y = average.gyro_Y;
reference.gyro_Z = average.gyro_Z;
reference.accel_X = average.accel_X;
reference.accel_Y = average.accel_Y;
reference.accel_Z = average.accel_Z;

config_receiver();
flush_rx_fifo();

T1CONbits.TON = 1;   //turn timer1 on
T4CONbits.TON = 1;  //turn timer4 on

//////////////////MAIN LOOP//////////////////

while(!end)
{
       while(!new_cycle);  //new cycle starts on timer1 interrupt
       new_cycle = 0;

       DMA1CONbits.CHEN=1; // Enable DMA
       AD1CON1bits.ADON = 1;       // turn ADC on
       while(!ADC_Done){ }  // wait for ADC to complete
       ADC_Done = 0;
       direction = 0;

       batt_voltage_check();       //BATTERY VOLTAGE CHECK

       //TRANSCEIVER CODE...
```

```
if(is_data_received())
{
	clear_status_flag(0x40);
	receive_data(payload_buffer);
	payload_buffer_to_dir(payload_buffer, &direction);
	__builtin_btg((unsigned int *)&LATG, 13);
	flush_rx_fifo();
}
if(seconds%patt_hold_time==1 && master_pattern_on_off == 1)
	pattern_on_off = 1;
//INPUT COMMAND
if(seconds%patt_hold_time==0 && pattern_on_off == 1)
{
	next_pattern();
	pattern_on_off = 0;
}
commands(direction);

//CONTROLS

error.gyro_X = current.gyro_X - reference.gyro_X;
error.gyro_Y = current.gyro_Y - reference.gyro_Y;
error.gyro_Z = current.gyro_Z - reference.gyro_Z;

error.accel_X = current.accel_X - reference.accel_X;
error.accel_Y = current.accel_Y - reference.accel_Y;
error.accel_Z = current.accel_Z - reference.accel_Z;

pitch_cont_input = pitch.Kp*error.accel_X - pitch.Kd*error.gyro_Y;
roll_cont_input =  roll.Kp*error.accel_Y + roll.Kd*error.gyro_X;
yaw_cont_input =   yaw.Kp*error.accel_Z - yaw.Kd*error.gyro_Z;

//pitch control
powerfront = half_max_power-pitch_cont_input;
powerback = half_max_power+pitch_cont_input;

//roll control
powerleft = half_max_power+roll_cont_input;
powerright = half_max_power-roll_cont_input;

//yaw control
powerFL_BR = half_max_power+yaw_cont_input;
powerFR_BL = half_max_power-yaw_cont_input;

powerFL = (powerleft + powerfront + powerFL_BR)/3;
powerFR = (powerright + powerfront + powerFR_BL)/3;
powerBL = (powerleft + powerback + powerFR_BL)/3;
powerBR = (powerright + powerback + powerFL_BR)/3;

//set saturation limits
saturate_at_limits(&powerFL, 0, full_max_power-1);
saturate_at_limits(&powerFR, 0, full_max_power-1);
saturate_at_limits(&powerBL, 0, full_max_power-1);
saturate_at_limits(&powerBR, 0, full_max_power-1);

//fronts connected to PWM
//back connected to CP
setPWM(powerFL, powerFR, powerBL, powerBR);
```

```
                if(logging_on_off)
                {
                        log_data(Data_Array);
                }

                if(end)
                {
                        transmit_log(Data_Array, payload_buffer);
                }

                PORTGbits.RG1 = 1;
        }

/* Turn off all LEDs to prepare for warning toggling */
        PORTGbits.RG1 = 1;
        PORTGbits.RG13 = 1;
        PORTFbits.RF0 = 1;
        PORTGbits.RG0 = 1;
        PORTGbits.RG12 = 1;

        while(1)
        {
                Delay( Delay_100mS_Cnt );
                Delay( Delay_100mS_Cnt );
                Delay( Delay_100mS_Cnt );
                Delay( Delay_100mS_Cnt );
                Delay( Delay_100mS_Cnt );

                /* Toggle all LEDs to warn that no batt voltage checking is being
done (since ADC's are off). */
                /* If device is left on and battery voltage drops, hardware could be
damaged */
                __builtin_btg((unsigned int *)&LATG, 1);
                __builtin_btg((unsigned int *)&LATG, 13);
                __builtin_btg((unsigned int *)&LATF, 0);
                __builtin_btg((unsigned int *)&LATG, 0);
                __builtin_btg((unsigned int *)&LATG, 12);
        }

}


int convolution(FilterData * object, const int new_element)
{
        object->inputstream[object->index] = new_element; //add new element to
array

        object->output = 0;
        int temp_index = object->index;
        int index_taps = 0;

        do{
                temp_index++; //start on oldest element
                if(temp_index >= BLOCK_LENGTH)
                {
                        temp_index = 0;      //wrap around (circular array)
                }
```

```
                object->output += object->inputstream[temp_index]*taps[index_taps];
//convolution
                index_taps++;
        }while(temp_index != object->index); //exit after newest element is used by
convolution

        object->index++;
        if(object->index >= BLOCK_LENGTH)
        {
                object->index = 0; //wrap around (circular array)
        }
        return (object->output)/10000;
}

void setPWM(int FrontLeft, int FrontRight, int RearLeft, int RearRight)
{
        OC1RS = RearLeft;    //R
        OC2R  = RearLeft;

        OC3RS = RearRight;   //RJ1
        OC4R  = RearRight;   //RJ2

        OC5R  = FrontLeft;   //FJ3
        OC6RS = FrontLeft;   //FJ4

        OC7R  = FrontRight;  //FJ1
        OC8RS = FrontRight;  //FJ2
}

void __attribute__((__interrupt__)) _DMA1Interrupt(void)
{
        AD1CON1bits.ADON = 0;       // turn ADC off

        if(filter_on_off == 1)
        {
                current.Batt_voltage = BufferA[0];
                current.Voltage33 = BufferA[1];
                current.VDD_Voltage = BufferA[2];
                current.gyro_X = convolution(&gyro_X_data, BufferA[3]);
                current.gyro_Y = convolution(&gyro_Y_data, BufferA[4]);
                current.gyro_Z = convolution(&gyro_Z_data, BufferA[5]);
                current.accel_Z = convolution(&accel_Z_data, BufferA[6]);
                current.accel_Y = convolution(&accel_Y_data, BufferA[7]);
                current.accel_X = convolution(&accel_X_data, BufferA[8]);
        }
        else
        {
                current.Batt_voltage = BufferA[0];
                current.Voltage33 = BufferA[1];
                current.VDD_Voltage = BufferA[2];
                current.gyro_X = BufferA[3];
                current.gyro_Y = BufferA[4];
                current.gyro_Z = BufferA[5];
                current.accel_Z = BufferA[6];
                current.accel_Y = BufferA[7];
                current.accel_X = BufferA[8];
        }
        ADC_Done = 1;
```

```
	IFS0bits.DMA1IF = 0; //Clear the DMA1 Interrupt Flag
}

void __attribute__((__interrupt__)) _T1Interrupt( void )
{
	PORTGbits.RG1 = 0;
	new_cycle = 1;

	IFS0bits.T1IF = 0;
}

void __attribute__((__interrupt__)) _T4Interrupt( void )
{
	__builtin_btg((unsigned int *)&LATG, 12);

	milliseconds++;
	if(milliseconds >= 1000)
	{
		milliseconds = 0;
		seconds++;
	}

	IFS1bits.T4IF = 0;
}

void calibrate(void)
{
	int i = 0;
	int j = 0;
	const int cali_iter = 50;
	double aver[9] = { 0 };

	while(i < cali_iter) // Repeat continuously
	{
		DMA1CONbits.CHEN = 1; // Enable DMA
		AD1CON1bits.ADON = 1;	// turn ADC on
		while(!(ADC_Done)){ }	// wait for ADC to complete
		ADC_Done = 0;
		for(j = 0; j<9 ; j++)
		{
			aver[j] += BufferA[j];
		}
		i++;
	}

	for(j = 0; j < 9; j++)
	{
		aver[j] = aver[j]/cali_iter;
	}

	average.Batt_voltage = aver[0];
	average.Voltage33 = aver[1];
	average.VDD_Voltage = aver[2];
	average.gyro_X = aver[3];
	average.gyro_Y = aver[4];
	average.gyro_Z = aver[5];
	average.accel_Z = aver[6];
	average.accel_Y = aver[7];
```

```
	average.accel_X = aver[8];
}

void commands(char direction)
{
	switch(direction)
	{
		case '=':
		half_max_power += 50;
		saturate_at_limits(&half_max_power, 0, 500);
		break;

		case '-':
		half_max_power -= 50;
		saturate_at_limits(&half_max_power, 0, 500);
		break;

		case 'i':
		yaw.Kp += step_size;
		saturate_at_limits_double(&yaw.Kp, 0, 100);
		break;

		case 'k':
		yaw.Kp -= step_size;
		saturate_at_limits_double(&yaw.Kp, 0, 100);
		break;

		case 'l':
		yaw.Kd += step_size;
		saturate_at_limits_double(&yaw.Kd, 0, 100);
		break;

		case 'j':
		yaw.Kd -= step_size;
		saturate_at_limits_double(&yaw.Kd, 0, 100);
		break;

		case 't':
		roll.Kp += step_size;
		saturate_at_limits_double(&roll.Kp, 0, 100);
		break;

		case 'g':
		roll.Kp -= step_size;
		saturate_at_limits_double(&roll.Kp, 0, 100);
		break;

		case 'h':
		roll.Kd += step_size;
		saturate_at_limits_double(&roll.Kd, 0, 100);
		break;

		case 'f':
		roll.Kd -= step_size;
		saturate_at_limits_double(&roll.Kd, 0, 100);
		break;

		case 'w':
```

```
pitch.Kp += step_size;
saturate_at_limits_double(&pitch.Kp, 0, 100);
break;

case 's':
pitch.Kp -= step_size;
saturate_at_limits_double(&pitch.Kp, 0, 100);
break;

case 'a':
pitch.Kd -= step_size;
saturate_at_limits_double(&pitch.Kd, 0, 100);
break;

case 'd':
pitch.Kd += step_size;
saturate_at_limits_double(&pitch.Kd, 0, 100);
break;

case '8':
reference.accel_X -= 50;
saturate_at_limits(&reference.accel_X, 0, 4095);
break;

case '5':
reference.accel_X += 50;
saturate_at_limits(&reference.accel_X, 0, 4095);
break;

case '4':
reference.accel_Y -= 50;
saturate_at_limits(&reference.accel_Y, 0, 4095);
break;

case '6':
reference.accel_Y += 50;
saturate_at_limits(&reference.accel_Y, 0, 4095);
break;

case '1':
reference.accel_Z -= 50;
saturate_at_limits(&reference.accel_Z, 0, 4095);
break;

case '3':
reference.accel_Z += 50;
saturate_at_limits(&reference.accel_Z, 0, 4095);
break;

case '2':
reference.accel_Z = average.accel_Z;
break;

case '7':
reference.accel_Y = average.accel_Y;
break;

case '9':
```

```
reference.accel_X = average.accel_X;
break;

case 'x':
OC1CONbits.OCM = 0b000; // Disable Output Compare Module
OC2CONbits.OCM = 0b000; // Disable Output Compare Module
OC3CONbits.OCM = 0b000; // Disable Output Compare Module
OC4CONbits.OCM = 0b000; // Disable Output Compare Module
OC5CONbits.OCM = 0b000; // Disable Output Compare Module
OC6CONbits.OCM = 0b000; // Disable Output Compare Module
OC7CONbits.OCM = 0b000; // Disable Output Compare Module
OC8CONbits.OCM = 0b000; // Disable Output Compare Module
break;

case 'v':
       if(logging_on_off == 1)
       {
              logging_on_off = 0;
       }
       else
       {
              logging_on_off = 1;
       }
break;

case 'c':
       if(filter_on_off == 1)
       {
              filter_on_off = 0;
       }
       else
       {
              filter_on_off = 1;
       }
break;

case 'b':
       if(master_pattern_on_off == 1)
       {
              master_pattern_on_off = 0;
              reference.accel_Z = average.accel_Z;
              reference.accel_Y = average.accel_Y;
              reference.accel_X = average.accel_X;
       }
       else
       {
              master_pattern_on_off = 1;
       }
break;

case 'z':
OC1CONbits.OCM = 0b000; // Disable Output Compare Module
OC2CONbits.OCM = 0b000; // Disable Output Compare Module
OC3CONbits.OCM = 0b000; // Disable Output Compare Module
OC4CONbits.OCM = 0b000; // Disable Output Compare Module
OC5CONbits.OCM = 0b000; // Disable Output Compare Module
OC6CONbits.OCM = 0b000; // Disable Output Compare Module
OC7CONbits.OCM = 0b000; // Disable Output Compare Module
```

```
                OC8CONbits.OCM = 0b000; // Disable Output Compare Module
                end = 1;
                break;
        }
}

void next_pattern()
{
        int i = 0;
        switch(pattern_num)
        {
                case 0:
                        for(i = 0; i<command_mult; i++)
                                commands('8');
                break;
                case 1:
                        for(i = 0; i<command_mult; i++)
                                commands('9');
                break;
                case 2:
                        for(i = 0; i<command_mult; i++)
                                commands('5');
                break;
                case 3:
                        for(i = 0; i<command_mult; i++)
                                commands('9');
                break;
                case 4:
                        for(i = 0; i<command_mult; i++)
                                commands('6');
                break;
                case 5:
                        for(i = 0; i<command_mult; i++)
                                commands('7');
                break;
                case 6:
                        for(i = 0; i<command_mult; i++)
                                commands('4');
                break;
                case 7:
                        for(i = 0; i<command_mult; i++)
                                commands('7');
                break;
                case 8:
                        for(i = 0; i<command_mult; i++)
                                commands('1');
                break;
                case 9:
                        for(i = 0; i<command_mult; i++)
                                commands('2');
                break;
                case 10:
                        for(i = 0; i<command_mult; i++)
                                commands('3');
                break;
                case 11:
                        for(i = 0; i<command_mult; i++)
                                commands('2');
```

```
			break;
		}
		pattern_num++;
		if(pattern_num > 11)
		{
			pattern_num = 0;
		}
}

void batt_voltage_check()
{
		if(current.Batt_voltage <= 2800)
		{
			PORTFbits.RF0 = 0;
		}
		else
		{
			PORTFbits.RF0 = 1;
		}
}

void transmit_log(const unsigned int * buff, char * payload_buffer)
{
		/* Disable interrupts to increase speed of transmission */
		IEC0bits.T1IE = 0;
		IEC0bits.DMA1IE = 0;
		IEC1bits.T4IE = 0;

		PORTGbits.RG0 = 1;
		PORTGbits.RG12 = 1;

		config_transmitter();
		unsigned int i;
		unsigned int values[6] = {0}; //can have up to 6 values
		send_ready_to_MATLAB(payload_buffer);
		for(i=0; i<cycle_num; i++)
		{
			values[0] = buff[packet_size*i];
			values[1] = buff[packet_size*i+1];
			values[2] = buff[packet_size*i+2];
			int_to_payload_buffer_MATLAB(payload_buffer, values, 3, 0);
			transmit(payload_buffer);   //transmit buffer

			values[0] = buff[packet_size*i+3];
			values[1] = buff[packet_size*i+4];
			values[2] = buff[packet_size*i+5];
			int_to_payload_buffer_MATLAB(payload_buffer, values, 3, 0);
			transmit(payload_buffer);   //transmit buffer

			values[0] = buff[packet_size*i+6];
			values[1] = buff[packet_size*i+7];
			values[2] = buff[packet_size*i+8];
			int_to_payload_buffer_MATLAB(payload_buffer, values, 3, 0);
			transmit(payload_buffer);   //transmit buffer

			values[0] = buff[packet_size*i+9];
			values[1] = buff[packet_size*i+10];
			values[2] = buff[packet_size*i+11];
```

```
                values[3] = buff[packet_size*i+12];
                values[4] = buff[packet_size*i+13];
                int_to_payload_buffer_MATLAB(payload_buffer, values, 5, 0);
                transmit(payload_buffer);   //transmit buffer

                values[0] = buff[packet_size*i+14];
                values[1] = buff[packet_size*i+15];
                values[2] = buff[packet_size*i+16];
                values[3] = buff[packet_size*i+17];
                values[4] = buff[packet_size*i+18];
                int_to_payload_buffer_MATLAB(payload_buffer, values, 5, 1);
                transmit(payload_buffer);   //transmit buffer

                __builtin_btg((unsigned int *)&LATG, 0);
        }
        send_finished_to_MATLAB(payload_buffer);
        PORTGbits.RG0 = 0;
}

void saturate_at_limits(int * value, int low, int hi)
{
        if(*value <= low)
                *value = low;
        else if (*value >= hi)
                *value = hi;
}

void saturate_at_limits_double(double * value, double low, double hi)
{
        if(*value <= low)
                *value = low;
        else if (*value >= hi)
                *value = hi;
}

void log_data(unsigned int * buff)
{
        if(cycle_num < max_cycle_num)
        {
                buff[packet_size*cycle_num]   = current.gyro_X;
                buff[packet_size*cycle_num+1] = current.gyro_Y;
                buff[packet_size*cycle_num+2] = current.gyro_Z;
                buff[packet_size*cycle_num+3] = current.accel_X;
                buff[packet_size*cycle_num+4] = current.accel_Y;
                buff[packet_size*cycle_num+5] = current.accel_Z;
                buff[packet_size*cycle_num+6] = current.Batt_voltage;
                buff[packet_size*cycle_num+7] = seconds;
                buff[packet_size*cycle_num+8] = milliseconds;
                buff[packet_size*cycle_num+9] = reference.gyro_X;
                buff[packet_size*cycle_num+10] = reference.gyro_Y;
                buff[packet_size*cycle_num+11] = reference.gyro_Z;
                buff[packet_size*cycle_num+12] = reference.accel_X;
                buff[packet_size*cycle_num+13] = reference.accel_Y;
                buff[packet_size*cycle_num+14] = powerfront;
                buff[packet_size*cycle_num+15] = powerleft;
                buff[packet_size*cycle_num+16] = powerFL_BR;
                buff[packet_size*cycle_num+17] = half_max_power;
                buff[packet_size*cycle_num+18] = cycle_num;
```

```
		cycle_num++;
	}
	else
	{
		PORTGbits.RG0 = 0;
	}
}



/*****************************************************************
* FileName:        init_ADC.c
* Dependencies:    p33FJ256GP710A.h
* Processor:       dsPIC33F
* Compiler:        MPLAB® C30 v2.01 or higher
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Author            Date      Comments on this revision
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* David Smith        2011       ADC module initialization
*****************************************************************/

#include "p33FJ256GP710A.h"

void Init_ADC_multi_input( void )
{
	AD1CON1bits.AD12B = 1;      // Select 12-bit mode, 1-channel mode
	AD1CON2bits.CHPS = 0;       // Select 0-channel mode
	AD1CON1bits.SIMSAM = 0; // Disable Simultaneous Sampling
	AD1CON2bits.ALTS = 0;       // Enable Alternate Input Selection
	AD1CON2bits.SMPI = 8;       // Select 9 conversions between interrupt (N-1)
	AD1CON2bits.VCFG = 0;       // use AVDD and AVSS as reference voltages
	AD1CON2bits.CSCNA = 1;      // Enable Channel Scanning
	AD1CON2bits.BUFM = 0;       // Always start at beginning of buffer after
interrupt
	AD1CON2bits.ALTS = 0;
	AD1CON1bits.ASAM = 1;       // Enable Automatic Sampling
	AD1CON1bits.SSRC = 0b111;   // internal counter generates SOC trigger
	AD1CON1bits.FORM = 0;       //integer
	AD1CON1bits.ADSIDL = 1;     //idle mode
	AD1CON3bits.SAMC = 31;
	AD1CON3bits.ADRC = 1;       //we are using internal clock 7.37MHz

	AD1CON1bits.ADDMABM = 1;
	IFS0bits.AD1IF = 0; // Clear the A/D interrupt flag bit
	IEC0bits.AD1IE = 0; // Do not Enable A/D interrupt

	AD1PCFGHbits.PCFG16 = 0;    //set as analog input
	AD1PCFGHbits.PCFG17 = 0;    //set as analog input
	AD1PCFGHbits.PCFG18 = 0;    //set as analog input
	AD1PCFGHbits.PCFG22 = 0;    //set as analog input
	AD1PCFGHbits.PCFG23 = 0;    //set as analog input
	AD1PCFGHbits.PCFG24 = 0;    //set as analog input
	AD1PCFGHbits.PCFG25 = 0;    //set as analog input
	AD1PCFGHbits.PCFG26 = 0;    //set as analog input
	AD1PCFGHbits.PCFG27 = 0;    //set as analog input

	// These will scan in ascending order
```

```
	// no matter what order they are listed below
	AD1CSSHbits.CSS16=1; // Enable AN16 for scan
	AD1CSSHbits.CSS17=1; // Enable AN17 for scan
	AD1CSSHbits.CSS18=1; // Enable AN18 for scan
	AD1CSSHbits.CSS22=1; // Enable AN22 for scan
	AD1CSSHbits.CSS23=1; // Enable AN23 for scan
	AD1CSSHbits.CSS24=1; // Enable AN24 for scan
	AD1CSSHbits.CSS25=1; // Enable AN25 for scan
	AD1CSSHbits.CSS26=1; // Enable AN26 for scan
	AD1CSSHbits.CSS27=1; // Enable AN27 for scan
}



/*********************************************************************
* FileName:        init_DMA1.c
* Dependencies:    p33FJ256GP710.h
* Processor:       dsPIC33F
* Compiler:        MPLAB® C30 v2.01 or higher
* REVISION HISTORY:
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Author            Date      Comments on this revision
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* David Smith               2011     DMA CH1 module initialization
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* ADDITIONAL NOTES:
*             I use DMA1 because its interrupt is lower priority than ADC1
*      which is useful in the program.
*********************************************************************/

#include "p33FJ256GP710A.h"

void initDma1(void)
{

	DMA1CONbits.AMODE = 0; // Configure DMA for Register indirect mode
                                                // with post-increment
	DMA1CONbits.MODE = 1; // Configure DMA for Single-Shot NOT Ping-Pong mode
	DMA1CONbits.SIZE = 0; // transfer word b/c 10-bit data
	DMA1PAD = (volatile unsigned int)&ADC1BUF0; // Point DMA to ADC1BUF0
	DMA1CNT = 8; // DMA request after 9 transfers (N-1)
	DMA1REQ = 13; // Select ADC1 as DMA Request source

	IFS0bits.DMA1IF = 0; //Clear the DMA interrupt flag bit
	IEC0bits.DMA1IE = 1; //Set the DMA interrupt enable bit
//	DMA1CONbits.CHEN=1; // Enable DMA
}



/*********************************************************************
* FileName:        init_PWM.c
* Dependencies:    p33FJ256GP710.h
* Processor:       dsPIC33F
* Compiler:        MPLAB® C30 v2.01 or higher
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Author            Date      Comments on this revision
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
* David Smith        2011     PWM module initialization
*****************************************************************/

#include "p33FJ256GP710A.h"

void init_PWM(void)
{
       TRISDbits.TRISD0 = 0;
       TRISDbits.TRISD1 = 0;
       TRISDbits.TRISD2 = 0;
       TRISDbits.TRISD3 = 0;
       TRISDbits.TRISD4 = 0;
       TRISDbits.TRISD5 = 0;
       TRISDbits.TRISD6 = 0;
       TRISDbits.TRISD7 = 0;

       //Initialize OC1
       OC1CONbits.OCM = 0b000; // Disable Output Compare Module
       OC1R = 100; // Write the duty cycle for the first PWM pulse
       OC1RS = 100; // Write the duty cycle for the second PWM pulse
       OC1CONbits.OCTSEL = 0; // Select Timer 2 as output compare time base
       OC1CONbits.OCM = 0b110; // Select the Output Compare mode

       //Initialize OC2
       OC2CONbits.OCM = 0b000; // Disable Output Compare Module
       OC2R = 100; // Write the duty cycle for the first PWM pulse
       OC2RS = 500; // Write the duty cycle for the second PWM pulse
       OC2CONbits.OCTSEL = 0; // Select Timer 2 as output compare time base
       OC2CONbits.OCM = 0b101; // Select the Output Compare mode

       //Initialize OC3
       OC3CONbits.OCM = 0b000; // Disable Output Compare Module
       OC3R = 100; // Write the duty cycle for the first PWM pulse
       OC3RS = 100; // Write the duty cycle for the second PWM pulse
       OC3CONbits.OCTSEL = 0; // Select Timer 2 as output compare time base
       OC3CONbits.OCM = 0b110; // Select the Output Compare mode

       //Initialize OC4
       OC4CONbits.OCM = 0b000; // Disable Output Compare Module
       OC4R = 100; // Write the duty cycle for the first PWM pulse
       OC4RS = 500; // Write the duty cycle for the second PWM pulse
       OC4CONbits.OCTSEL = 0; // Select Timer 2 as output compare time base
       OC4CONbits.OCM = 0b101; // Select the Output Compare mode

       //Initialize OC5
       OC5CONbits.OCM = 0b000; // Disable Output Compare Module
       OC5R = 100; // Write the duty cycle for the first PWM pulse
       OC5RS = 500; // Write the duty cycle for the second PWM pulse
       OC5CONbits.OCTSEL = 0; // Select Timer 2 as output compare time base
       OC5CONbits.OCM = 0b101; // Select the Output Compare mode

       //Initialize OC6
       OC6CONbits.OCM = 0b000; // Disable Output Compare Module
       OC6R = 100; // Write the duty cycle for the first PWM pulse
       OC6RS = 100; // Write the duty cycle for the second PWM pulse
       OC6CONbits.OCTSEL = 0; // Select Timer 2 as output compare time base
       OC6CONbits.OCM = 0b110; // Select the Output Compare mode
```

```
        //Initialize OC7
        OC7CONbits.OCM = 0b000; // Disable Output Compare Module
        OC7R = 100; // Write the duty cycle for the first PWM pulse
        OC7RS = 500; // Write the duty cycle for the second PWM pulse
        OC7CONbits.OCTSEL = 0; // Select Timer 2 as output compare time base
        OC7CONbits.OCM = 0b101; // Select the Output Compare mode

        //Initialize OC8
        OC8CONbits.OCM = 0b000; // Disable Output Compare Module
        OC8R = 100; // Write the duty cycle for the first PWM pulse
        OC8RS = 100; // Write the duty cycle for the second PWM pulse
        OC8CONbits.OCTSEL = 0; // Select Timer 2 as output compare time base
        OC8CONbits.OCM = 0b110; // Select the Output Compare mode

        // Initialize and enable Timer2
        T2CONbits.TON = 0; // Disable Timer
        T2CONbits.TCS = 0; // Select internal instruction cycle clock
        T2CONbits.TGATE = 0; // Disable Gated Timer mode
        T2CONbits.TCKPS = 0b00; // Select 1:1 Prescaler
        TMR2 = 0x00; // Clear timer register
        PR2 = 500; // Load the period value
        IPC1bits.T2IP = 0x01; // Set Timer 2 Interrupt Priority Level
        IFS0bits.T2IF = 0; // Clear Timer 2 Interrupt Flag
        IEC0bits.T2IE = 0; // Enable Timer 2 interrupt
        T2CONbits.TON = 1; // Start Timer
}



/************************************************************************
* FileName:        init_SPI.c
* Dependencies:    p33FJ256GP710A.h
* Processor:       dsPIC33F
* Compiler:        MPLAB® C30 v2.01 or higher
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Author            Date      Comments on this revision
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* David Smith       2011      SPI module initialization
************************************************************************/

#include "p33FJ256GP710A.h"

void init_SPI( void )
{

        IFS0bits.SPI1IF = 0; // Clear the Interrupt Flag
        IEC0bits.SPI1IE = 0; // Disable the Interrupt

        // SPI1CON1 Register Settings
        SPI1CON1bits.DISSCK = 0; // Internal Serial Clock is Enabled
        SPI1CON1bits.DISSDO = 0; // SDOx pin is controlled by the module
        SPI1CON1bits.MODE16 = 0; // Communication is byte-wide (8 bits)
        SPI1CON1bits.MSTEN = 1; // Master mode Enabled
        SPI1CON1bits.SMP = 0; // Input data is sampled at the middle of data
        // output time
        SPI1CON1bits.CKE = 1; // Serial output data changes on transition
        // from Idle clock state to active clock state
        SPI1CON1bits.CKP = 0; // Idle state for clock is a low level;
```

```
        // active state is a high level
        SPI1CON1bits.SSEN = 0;      //SS1 controlled by port, not module
        SPI1STATbits.SPIEN = 1; // Enable SPI module

        IFS0bits.SPI1IF = 0; // Clear the Interrupt Flag
        IEC0bits.SPI1IE = 0; // Disable the Interrupt

}



/*********************************************************************
* FileName:        init_timer1.c
* Dependencies:    p33FJ256GP710.h
* Processor:       dsPIC33F
* Compiler:        MPLAB® C30 v2.01 or higher
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Author            Date      Comments on this revision
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Richard Fischer   07/14/05  Initialization of Timer1 for RTC demo
* David Smith        2011     Adapted for MAV
*********************************************************************/

#include "p33FJ256GP710A.h"

void Init_Timer1( void )
{
        T1CONbits.TON = 0;   //disable timer

        /* ensure Timer 1 is in reset state */
        T1CON = 0;

        /* reset Timer 1 interrupt flag */
        IFS0bits.T1IF = 0;

        /* set Timer1 interrupt priority level to 4 */
        IPC0bits.T1IP = 4;

        /* enable Timer 1 interrupt */
        IEC0bits.T1IE = 1;

        T1CONbits.TCKPS = 0b10;

        /* set Timer 1 period register */
//      PR1 = 0x7FFF; //about 2 HZ
//      PR1 = 0x666; // 7 HZ
//      PR1 = 0xCCC; // 60 ms period allows for wireless receiving.
        PR1 = 0x333; // maxs out loop frequency

        /* select internal timer clock Fosc/2 */
        T1CONbits.TCS = 0;

        TMR1 = 0x00;
        // timer still needs to be turned on using
        // T1CONbits.TON = 1; (can be done in main code)
}
```

```
/*********************************************************************
* FileName:        init_timer4.c
* Dependencies:    p33FJ256GP710.h
* Processor:       dsPIC33F
* Compiler:        MPLAB® C30 v2.01 or higher
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Author            Date      Comments on this revision
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Richard Fischer   07/14/05  Initialization of Timer1 for RTC demo
* David Smith        2011     Adapted for MAV
*********************************************************************/

#include "p33FJ256GP710A.h"

void Init_Timer4( void )
{
       T4CONbits.TON = 0;   //disable timer

       /* ensure Timer 1 is in reset state */
       T4CON = 0;

       /* reset Timer 4 interrupt flag */
       IFS1bits.T4IF = 0;

       /* set Timer4 interrupt priority level to 4 */
       IPC6bits.T4IP = 5;

       /* enable Timer 4 interrupt */
       IEC1bits.T4IE = 1;

       T4CONbits.TCKPS = 0b01;

       T4CONbits.T32 = 0;

       T4CONbits.TCS = 0; //Internal Clock, Fosc/2

       TMR4 = 0x00;

       /* set Timer 4 period register */
       PR4 = 460;    // 460 counts per millisec

       // timer still needs to be turned on using
       // T4CONbits.TON = 1; (can be done in main code)
}



/*********************************************************************
* FileName:        xceiver_init.h
* Dependencies:    p33FJ256GP710A.h
* Processor:       dsPIC33F
* Compiler:        MPLAB® C30 v2.01 or higher
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Author            Date      Comments on this revision
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* David Smith        2011     Config functions for nRF24L01+
*********************************************************************/
```

```
char SPI_read_write_byte(char data);
void init_transceiver( void );
void config_transmitter(void);
void config_receiver(void);
void transmit(char *buf_ptr);
void retransmit(void);
void receive_data(char *buf_ptr);
char read_status_flag( void );
void clear_status_flag(char value);
void flush_rx_fifo(void);
char is_data_received(void);
void cs_assert(void);
void cs_disassert(void);
void ce_assert(void);
void ce_disassert(void);
void send_ready_to_MATLAB(char * buffer);
void send_finished_to_MATLAB(char * buffer);
void payload_buffer_to_dir(char* buffer, char * dir);
void int_to_payload_buffer_actual_values(char* buffer, const int * int_array);
void payload_buffer_to_int_actual_values(const char* buffer, int * int_array);
void int_to_payload_buffer_MATLAB(char* buffer, const unsigned int * int_array,
char array_size, char endline);
void int_to_payload_buffer(char* buffer, const unsigned int * int_array, char
array_size, char option123);
//array_size should not be > 5!
void payload_buffer_to_int_chars(const char* buffer, unsigned int * int_array,
char array_size);
//array_size should not be > 5!
void flush_buffer(char * buff);



/*********************************************************************
* FileName:        xceiver_init.c
* Dependencies:    p33FJ256GP710A.h, xceiver_init.h
* Processor:       dsPIC33F
* Compiler:        MPLAB® C30 v2.01 or higher
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Author             Date      Comments on this revision
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* David Smith        2011     Implementation for nRF24L01+
*********************************************************************/

#include "p33FJ256GP710A.h"
#include "xceiver_init.h"
#include "delay.h"

void init_transceiver( void )
{
       TRISBbits.TRISB2 = 0;
       PORTBbits.RB2 = 1;
       TRISDbits.TRISD9 = 0;
       PORTDbits.RD9 = 0;

       // Setup CONFIG register

       cs_assert();
```

```
	SPI_read_write_byte(0x20);
	SPI_read_write_byte(0x79); // No interrupt used, CRC enabled, CRC encoding
1 byte, PRX mode
	cs_disassert();

	cs_assert();
	SPI_read_write_byte(0x21); //enable auto acknowledgement on data pipes 0 &
1
	SPI_read_write_byte(0x03);
	cs_disassert();

	cs_assert();
	SPI_read_write_byte(0x24); //setup auto retransmission
	SPI_read_write_byte(0x1F); //wait 500us, retransmit up to 15 times
	cs_disassert();

	cs_assert();
	SPI_read_write_byte(0x26);
	SPI_read_write_byte(0x07); //data rate: 1Mbps, RF power: 0dBm, LNA gain:
high
	cs_disassert();

	cs_assert();
	SPI_read_write_byte(0x25);
	SPI_read_write_byte(0x02); //RF channel setup
	cs_disassert();

	cs_assert();
	SPI_read_write_byte(0x31);
	SPI_read_write_byte(32); //static payload length
	cs_disassert();

	cs_assert();
	SPI_read_write_byte(0x23);
	SPI_read_write_byte(0x03); //address width 5 bytes
	cs_disassert();

	cs_assert();
	SPI_read_write_byte(0x2A);
	SPI_read_write_byte(0xE7+4); //RX address LSB
	SPI_read_write_byte(0xE7+3);
	SPI_read_write_byte(0xE7+2);
	SPI_read_write_byte(0xE7+1);
	SPI_read_write_byte(0xE7); //RX address MSB
	cs_disassert();

	cs_assert();
	SPI_read_write_byte(0x30);
	SPI_read_write_byte(0xE7+4); //TX address LSB
	SPI_read_write_byte(0xE7+3);
	SPI_read_write_byte(0xE7+2);
	SPI_read_write_byte(0xE7+1);
	SPI_read_write_byte(0xE7); //TX address MSB
	cs_disassert();
}

void config_transmitter(void)
{
```

```
	ce_disassert();

	cs_assert();
	SPI_read_write_byte(0x20);
	SPI_read_write_byte(0x7A); // No interrupt used, CRC enabled, CRC encoding
1 byte, PTX mode
	cs_disassert();
}

void config_receiver(void)
{
	cs_assert();
	SPI_read_write_byte(0x20); //write to config
	SPI_read_write_byte(0x7B); // No interrupt used, CRC enabled, CRC encoding
1 byte, PRX mode
	cs_disassert();

	ce_assert();
}

void transmit(char *buf_ptr)
{
	int i = 0;

	// Load TX FIFO
	cs_assert();
	SPI_read_write_byte(0b10100000); //W_TX_PAYLOAD
	for(i = 0; i<32; i++)
	{
		SPI_read_write_byte(*buf_ptr++); // set payload
	}
	cs_disassert();

	ce_assert();
	Delay_Us( Delay15uS_count );
	ce_disassert();

	while((!(read_status_flag() & 0x20)) && (!(read_status_flag() & 0x10)));

	while(read_status_flag() & 0x10)
		retransmit();

	clear_status_flag(0x20);
}

void retransmit(void)
{
	clear_status_flag(0x20);
	clear_status_flag(0x10);

	ce_assert();
	Delay_Us( Delay15uS_count );
	ce_disassert();

	while((!(read_status_flag() & 0x20)) && (!(read_status_flag() & 0x10)));
}

char SPI_read_write_byte(char data_byte)
```

```
{
	SPI1BUF = data_byte; // Write data to be transmitted
	while(!SPI1STATbits.SPIRBF);
	return SPI1BUF;
}

void receive_data(char *buf_ptr)
{
	int i = 0;

	flush_buffer(buf_ptr);

	ce_disassert();

	cs_assert();
	SPI_read_write_byte(0x61); // R_RX_PAYLOAD -> read payload
	for(i = 0; i<32; i++)
	{
		buf_ptr[i] = SPI_read_write_byte(0xFF);
	}
	cs_disassert();

	ce_assert();
}

char read_status_flag( void )
{
	char status;

	cs_assert();
	status = SPI_read_write_byte(0xff); //NOP, to read status register
	cs_disassert();

	return status;
}

void clear_status_flag(char value)
{
	cs_assert();
	SPI_read_write_byte(0x27);
	SPI_read_write_byte(value); // clear flag
	cs_disassert();
}

void flush_rx_fifo(void)
{
	cs_assert();
	SPI_read_write_byte(0xe2);
	cs_disassert();
}

char is_data_received(void)
{
	if(read_status_flag() & 0x40)
	{
		return 1;
	}
	return 0;
```

```
}

void cs_assert(void)
{
	PORTBbits.RB2 = 0; //start packet
}

void cs_disassert(void)
{
	PORTBbits.RB2 = 1; //stop packet
	asm volatile("nop");
}

void ce_assert(void)
{
	PORTDbits.RD9 = 1;
}

void ce_disassert(void)
{
	PORTDbits.RD9 = 0;
}

void int_to_payload_buffer_actual_values(char* buffer, const int * int_array)
{
	int number_of_ints = 3; // do not make > 16
	int i;
	for(i = 0; i<number_of_ints; i++)
	{
		buffer[2*i] = (char)(int_array[i]%256);
		buffer[2*i+1] = (char)(int_array[i]>>8);
	}
}

void payload_buffer_to_int_actual_values(const char* buffer, int * int_array)
{
	int number_of_ints = 3; // do not make > 16
	int i;
	for(i = 0; i<number_of_ints; i++)
	{
		int_array[i] = ((int)(buffer[2*i]) & 0x00FF) +
((int)(buffer[2*i+1]))*256;
	}
}

void send_ready_to_MATLAB(char * buffer)
{
	flush_buffer(buffer);
	buffer[0] = 0x42;    //sends "B"
	transmit(buffer);    //transmit buffer
}

void send_finished_to_MATLAB(char * h_buffer)
{
	flush_buffer(h_buffer);
	h_buffer[0] = 0x47;  // sends "G"
	transmit(h_buffer);  //transmit buffer
}
```

```
void int_to_payload_buffer_MATLAB(char* buffer, const unsigned int * int_array,
char array_size, char endline)
{
	unsigned int temp[32+1], Char[5];
	int i, j;
	flush_buffer(buffer);
	for(i = 0; i<array_size; i++)
	{
		temp[4*i+3] = ((int_array[i]) & 0x000F);
		temp[4*i+2] = ((int_array[i]) & 0x00F0);
		temp[4*i+1] = ((int_array[i]) & 0x0F00);
		temp[4*i]   = ((int_array[i]) & 0xF000);

		for(j = 0; j<4; j++)
		{
			Char[j] = temp[4*i+3]%10 + temp[4*i+2]%10 + temp[4*i+1]%10 +
temp[4*i]%10;
			if(Char[j] >= 10)
			{
				if(Char[j] >= 20)
				{
					if(Char[j] >=30) //since value will be <4096,
char[j] will never be >30
					{
						temp[4*i] += 30;
					}
					else
					{
						temp[4*i] += 20;
					}
				}
				else
				{
					temp[4*i] += 10;
				}
				Char[j] = Char[j]%10;
			}

			temp[4*i+3] = temp[4*i+3]/10;
			temp[4*i+2] = temp[4*i+2]/10;
			temp[4*i+1] = temp[4*i+1]/10;
			temp[4*i]   = temp[4*i]/10;

			buffer[(5*(i+1))-j-2] = Char[j] + 0x30;
		}
		buffer[(5*(i+1))-1] = 0x20;  //space
	}
	buffer[5*(array_size)-1] = 0x20; //space
	if(endline)
	{
		buffer[(5*(array_size)-1)] = ';'; //semicolon
		buffer[(5*(array_size)-1)+1] = 0xA; //newline
		buffer[(5*(array_size)-1)+2] = 0xD; //carriage return
	}
}
```

```
void int_to_payload_buffer(char* buffer, const unsigned int * int_array, char
array_size, char option123)
{
	unsigned int temp[32+1], Char[5];
	int i, j;
	flush_buffer(buffer);
	for(i = 0; i<array_size; i++)
	{
		temp[4*i+3] = ((int_array[i]) & 0x000F);
		temp[4*i+2] = ((int_array[i]) & 0x00F0);
		temp[4*i+1] = ((int_array[i]) & 0x0F00);
		temp[4*i]   = ((int_array[i]) & 0xF000);

		for(j = 0; j<5; j++)
		{
			Char[j] = temp[4*i+3]%10 + temp[4*i+2]%10 + temp[4*i+1]%10 +
temp[4*i]%10;
			if(Char[j] >= 10)
			{
				Char[j] = Char[j]%10;
				temp[4*i] += 10;
			}

			temp[4*i+3] = temp[4*i+3]/10;
			temp[4*i+2] = temp[4*i+2]/10;
			temp[4*i+1] = temp[4*i+1]/10;
			temp[4*i]   = temp[4*i]/10;

			buffer[(6*(i+1))-j-2] = Char[j] + 0x30;
		}
		buffer[(6*i)-1] = 0x20;  //space
		switch(option123)
		{
			case 1:
				buffer[(6*i)] = 0x58 + i; //insert X, Y, Z
				break;
			case 2:
				buffer[(6*i)] = 0x55 + i; //insert U, V, W
				break;
			case 3:
				buffer[(6*i)] = 0x42 + i; //insert B, C, D
				buffer[(19)] = 0xA; //insert new line
				break;
		}
	}
}

void payload_buffer_to_dir(char* buffer, char * dir)
{
	*dir = buffer[0];
	flush_buffer(buffer);
}

void flush_buffer(char * buff)
{
	int i;
	for(i=0; i<32+1; i++)
	{
```

```
            buff[i] = 0;
        }
}
```

```
/*********************************************************************
 *                    Simple Delay Routines
 *********************************************************************
 * FileName:        delay.h
 * Dependencies:    delay.c
 * Processor:       dsPIC33F
 * Complier:        MPLAB C30 v2.01.00 or higher
 *
 * Author               Date    Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Richard Fischer      7/14/05 Initial release for LCD support
 * Priyabrata Sinha     1/27/06 Ported to non-prototype devices
 * David Smith                  2011      Adapted for MAV
 ********************************************************************/

#define Fcy  23031250

void Delay( unsigned int delay_count );
void Delay_Us( unsigned int delayUs_count );

#define Delay200uS_count  (Fcy * 0.0002) / 228
#define Delay15uS_count   (Fcy * 0.00001) / 228
//note, can't go much lower than 20us

#define Delay_1mS_Cnt         (Fcy * 0.001) / 3838
#define Delay_2mS_Cnt         (Fcy * 0.002) / 3838
#define Delay_5mS_Cnt         (Fcy * 0.005) / 3838
#define Delay_15mS_Cnt        (Fcy * 0.015) / 3838
#define Delay_1S_Cnt   (Fcy * 1) / 3838
#define Delay_100mS_Cnt   (Fcy * 0.1) / 3838
```

```
/*********************************************************************
 *                    Simple Delay Routines
 *********************************************************************
 * FileName:        delay.c
 * Dependencies:    delay.h
 * Processor:       dsPIC33F
 * Complier:        MPLAB C30 v2.01.00 or higher
 *
 * Author               Date    Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Richard Fischer      7/14/05 Initial release for LCD support
 * Priyabrata Sinha     1/27/06 Ported to non-prototype devices
 * David Smith                  2011      Adapted for MAV
 ********************************************************************/

#include "delay.h"

unsigned int temp_count;
```

```
void Delay( unsigned int delay_count )
{
	temp_count = delay_count +1;
	asm volatile("outer: dec _temp_count");
	asm volatile("cp0 _temp_count");
	asm volatile("bra z, done");
	asm volatile("do #300, inner" );
	asm volatile("nop");
	asm volatile("inner: nop");
	asm volatile("bra outer");
	asm volatile("done:");
}


void Delay_Us( unsigned int delayUs_count )
{
	temp_count = delayUs_count +1;
	asm volatile("outer1: dec _temp_count");
	asm volatile("cp0 _temp_count");
	asm volatile("bra z, done1");
	asm volatile("do #13, inner1" );
	asm volatile("nop");
	asm volatile("inner1: nop");
	asm volatile("bra outer1");
	asm volatile("done1:");
}
```

# REFERENCES


[1] ACKERMAN, E., "Aerovironment's nano hummingbird surveillance bot would probably fool you," March 2011.

[2] DOMINGUES, J., "Quadrotor prototype," Master's thesis, Instituto Superior Tecnico, 2009.

[3] HENRIQUES, B., "Estimation and control of a quadrotor attitude," Master's thesis, Instituto Superior Technico, 2011.

[4] HENSON, M. A. S. D. E., *Nonlinear Process Control*. Prentice-Hall, INC., 1997.

[5] KHAN, Z.A. AGRAWAL, S., "Control of longitudinal flight dynamics of a flapping-wing micro air vehicle using time-averaged model and differential flatness based controller," *American Control Conference*, vol. 9-13 July 2007, pp. 5284 – 5289, 2007.

[6] KOLDBAEK, C. M. L. C. T. S. K., "Modelling and control of autonomous quad-rotor." University of Aalborg, June 2010.

[7] RATTI, J., *QV: The Quad Winged, Energy Efficient, Six degree of Freedom Capable Micro Aerial Vehicle*. PhD thesis, Georgia Institute of Technology, 2011.

[8] SCHENATO, L., *Analysis and Control of Flapping Flight: from Biological to Robotic Insects*. PhD thesis, University of California Berkeley, 2003.